AD-A256 287

TATION PAGE

Form Approved
OMB No. 0704-0188

DATE

3. REPORT TYPE AND DATES COVERED
FINAL 1 May 90 - 30 Jun 92

4. TITLE AND SUBTITLE
"COMPUTATION & COMMUNICATION CONSTRAINTS FOR DISTRIBUTED ESTIMATION SYSTEMS" (U)

5. FUNDING NUMBERS
61102F
2304/A5

(2)

6. AUTHOR(S)
Dr. John A. Bubner

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dept of Electrical/Computer Engineering
University of Wisconsin
Madison WI 53706-1691

AFOSR-TR-

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR-90-0181

DTIC
ELECTE
OCT 01 1992
S A D

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
Distribution unlimited

12b. DISTRIBUTION CODE
UL

Research performed under this grant fall into four areas which are reflected in published papers: (1) J.A. Gubner, "Random Coding for the Constrained Multiple-Access Arbitrarily Varying Channel," Proc. Twenty-Eighth Annual Allerton Conf. Commun. Contr. Comput., Univ of Illinois, Urbana, IL, pp. 684-690, Oct. 1990, _____, "The Capacity Region of the Additive Multiple-Access Arbitrarily Varying Channel," Proc. 1991 IEEE Int. Symp. Inform. Theory, Budapest, Hungary, p. 218, June 1991, _____, "On the Capacity Region of the Discrete Additive Multiple-Access Arbitrarily Varying Channel," IEEE Trans. Inform. Theory, vol 38,no4,pp.1344-1347, July 1991 - concerned arbitrarily varying channels. (2) _____, "Distributed Estimation & Quantization," IEEE Trans. Inform. Theory, submitted 1992, and W.H. Wehbe, "Quantization for Distributed Estimation Systems," M.S. report, Dep,Elect.Comp.Eng., University of Wisconsin, Madison, 1992 focussed on the design of distributed estimation systems subject to communication and computation constraints. (3) _____, "On the Computation of Shot-Noise Probability Distributions," Trans.Inform.Theory, submitted 1992, and R.E. Sequeira, J.A. Gubner, and B.E.A. Saleh, "Image Detection Under Low-Level Illumination," IEEE Trans.Image Proc., in press, concerned the computation of shot-noise probability distributions and image detection based on shot-noise observations. (4) J.A. Gubner and W.B. Chang, "Wavelet Transofrms for Discrete-Time Periodic Signals," IEEE Trans.Signal Proc., submitted 1992 is a tutorial paper on wavelet transforms for discrete-time periodic signals.

14. SUBJECT TERMS

15. NUMBER OF PAGES
63

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Final Technical Report

AFOSR-90-0181

for the period 05/01/90 through 06/30/92

Computation and Communication Constraints for Distributed Estimation Systems

John A. Gubner, Assistant Professor, Principal Investigator

Department of Electrical and Computer Engineering
University of Wisconsin–Madison
Madison, WI 53706–1691

Phone: (608) 263–1471
e-mail: gubner@engr.wisc.edu

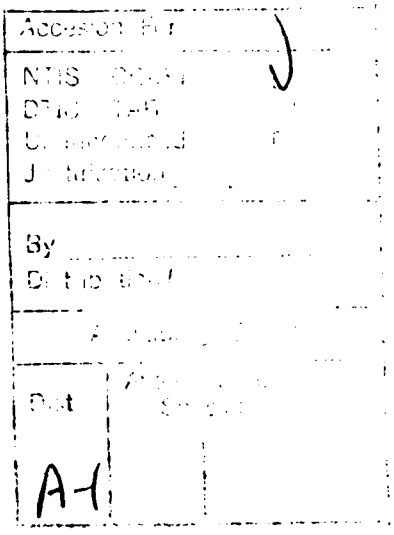


TABLE OF CONTENTS

SUMMARY OF WORK COMPLETED

Research performed under this grant contributed to papers [1]–[8] listed below in the references. The papers fall into four areas.

1. [1]–[3] concerned arbitrarily varying channels. Preliminary versions of [1] and [3] were included in last year's report. These papers have since appeared in the literature and are therefore not included in this report.

2. [4] and [8] focused on the design of distributed estimation systems subject to communication and computation constraints. The theory can be found in [4] and a description and listing of the programs used to implement the design algorithm can be found in [8]. Both of these papers are included with this report.

3. [5] and [7] concerned the computation of shot-noise probability distributions and image detection based on shot-noise observations. These papers are included with this report.

4. [6] is a tutorial paper on wavelet transforms for discrete-time periodic signals. This paper is included with this report.

REFERENCES

[1] J. A. Gubner, "Random coding for the constrained multiple-access arbitrarily varying channel," *Proc. Twenty-Eighth Annual Allerton Conf. Commun. Contr. Comput.*, University of Illinois, Urbana, IL, pp. 684-690, Oct. 1990.

[2] ——, "The capacity region of the additive multiple-access arbitrarily varying channel," *Proc. 1991 IEEE Int. Symp. Inform. Theory*, Budapest, Hungary, p. 218, June 1991.

[3] ——, "On the capacity region of the discrete additive multiple-access arbitrarily varying channel," *IEEE Trans. Inform. Theory*, vol. 38, no. 4, pp. 1344-1347, July 1992.

[4] ——, "Distributed estimation and quantization," *IEEE Trans. Inform. Theory*, submitted 1992.

[5] ——, "On the computation of shot-noise probability distributions," *IEEE Trans. Inform. Theory*, submitted 1992.

[6] J. A. Gubner and W.-B. Chang, "Wavelet transforms for discrete-time periodic signals," *IEEE Trans. Signal Proc.*, submitted 1992.

[7] R. E. Sequeira, J. A. Gubner, and B. E. A. Saleh, "Image detection under low-level illumination," *IEEE Trans. Image Proc.*, in press.

[8] W. H. Wehbe, "Quantization for distributed estimation systems," M.S. report, Dep. Elect. Comp. Eng., Univ. of Wisconsin, Madison, 1992.

Appendix A

Reprint of Reference [4]

Distributed Estimation and Quantization

John A. Gubner, *Member, IEEE*

Abstract — We develop an algorithm for the design of an n-sensor distributed estimation system subject to communication and computation constraints. The algorithm uses only bivariate probability distributions and yields locally optimal estimators that satisfy the required system constraints. It is shown that the algorithm is a generalization of the classical Lloyd-Max results.

Index Terms — Distributed estimation, quantization, Lloyd-Max algorithm.

I. INTRODUCTION

Consider the distributed estimation system shown in Fig. 1. The system consists of n sensor platforms whose respective measurements, $Y_1, \ldots, Y_n$, are related to some unobservable quantity, say X. Each sensor platform processes its respective measurement and transmits the result over a communication channel to a common fusion center. The sensors do not communicate with each other, and there is no feedback from the fusion center to the sensor platforms. The task of the fusion center is to estimate the unobserved quantity X. We denote this estimate by $\hat{X}$. Clearly, $\hat{X}$ is a function of $Y_1, \ldots, Y_n$, and we can write $\hat{X} = f(Y_1, \ldots, Y_n)$ for some function f. The problem then is to choose the function f so that $\hat{X}$ is close to X in some sense. For example, it is well known that in the appropriate probabilistic setting, the minimum-mean-square-error estimate of X given $Y_1, \ldots, Y_n$ is the conditional expectation of X given $Y_1, \ldots, Y_n$, denoted $\mathsf{E}[X \mid Y_1, \ldots, Y_n]$. However, there are many situations in which the conditional expectation does not provide a satisfactory solution to the problem of choosing f:

1) In general, the functional form of $\mathsf{E}[X \mid Y_1, \ldots, Y_n]$ as a function of $Y_1, \ldots, Y_n$ is difficult determine, and it requires knowledge of the joint probability distribution of $X, Y_1, \ldots, Y_n$. In practice this complete joint distribution may not be available.
2) To compute $\mathsf{E}[X \mid Y_1, \ldots, Y_n]$, the fusion center must in general have access to all of the sensor measurements $Y_1, \ldots, Y_n$. Hence, even if the sensor platforms have local processing capability, it is of little use in computing $\mathsf{E}[X \mid Y_1, \ldots, Y_n]$. If the number of sensor platforms is very large, the burden of computing $\mathsf{E}[X \mid Y_1, \ldots, Y_n]$ at the fusion center, even if the formula is relatively simple, may be prohibitive. Such considerations are important if the estimate of X must be computed in real time. By using a suboptimal estimator of X for which some of the processing can be done locally at the sensor platforms, it may be possible to design an acceptable estimator that can operate in real time.
3) As indicated in Fig. 1, the sensor platforms transmit their data to the fusion center. However, using any physical communication system, it is not possible to transmit real-valued quantities without distortion. In this situation, the conditional expectation, or even the best linear estimate, is generally a physically unrealizable solution.

In this paper we develop an algorithm to design solutions to the distributed estimation problem that do not suffer from these difficulties.

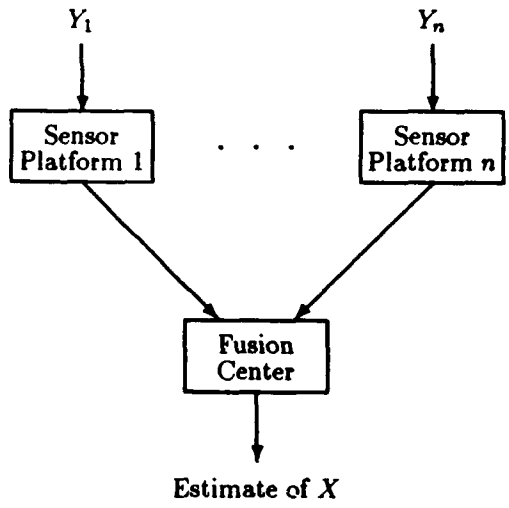


Fig. 1. A Distributed Estimation System.

II. BACKGROUND AND NOTATION

Our approach is to consider quantization for distributed estimation systems. The goal of quantization in such systems is to provide a good estimate of the unobservable, X, rather than to reconstruct the sensor measurements $Y_1, \ldots, Y_n$ as in [3]. Quantization for estimation has been studied for a single sensor by Ephraim and Gray [2] and by Ayanoglu [1]. The multi-sensor case has been studied by Lam and Reibman [5], and we discuss their work in

This research was presented in part at the 1990 IEEE International Symposium on Information Theory, San Diego, CA, Jan. 14-19, 1990. This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-90-0181.

The author is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.

more detail below. The paper by Zhang and Berger [9] considers an asymptotic estimation problem in which the observations are discrete random variables taking finitely many values and the unobservable quantity is not a random variable, but a deterministic and unknown parameter in some finite-dimensional Euclidean space.

A. System Model

Let $X, Y_1, \ldots, Y_n$ be real-valued random variables on some probability space $(\Omega, \mathcal{F}, \mathsf{P})$. Each sensor platform k processes its measurement Y_k to obtain an output Z_k. Each Z_k is then transmitted to the fusion center. We assume that the communication channel connecting the sensors to the fusion center has a positive capacity, and that the use of error-correcting codes permits us to view the channel as noiseless. We suppose that the channel can transmit messages of $\log_2 N$ bits without error, where $N \geq 2$ is an integer. For each k, let $A_{k1}, \ldots, A_{kN}$ be a partition of $\mathbb{R}$. We require that the sensor platform output Z_k be given by

$$Z_k \triangleq \sum_{i=1}^{N} (i-1) I_{A_{ki}}(Y_k). \tag{1}$$

where $I_A(y)$ denotes the indicator function of the set $A \subset \mathbb{R}$; i.e., $I_A(y) = 1$ if $y \in A$ and $I_A(y) = 0$ otherwise.

Under the preceding constraints, the function f discussed in Section I must be of the form

$$f(Y_1, \ldots, Y_n) = h(Z_1, \ldots, Z_n),$$

where each Z_k is equal to the function of Y_k determined by (1).

B. Relation to [5]

We now briefly summarize the approach in [5]. If the sets $\{A_{ki}\}$ are fixed, one wants to find a function $h(Z_1, \ldots, Z_n)$ that minimizes the mean-square error,

$$\mathsf{E}[|X - h(Z_1, \ldots, Z_n)|^2]; \tag{2}$$

hence, the optimal h is the conditional expectation,

$$h(z_1, \ldots, z_n) = \mathsf{E}[X \mid Z_1 = z_1, \ldots, Z_n = z_n].$$

If we set $i_1 = z_1 + 1, \ldots, i_n = z_n + 1$ and let

$$B \triangleq A_{1i_1} \times \cdots \times A_{ni_n},$$

then this conditional expectation is given by

$$\frac{\int_B \mathsf{E}[X \mid Y_1 = y_1, \ldots, Y_n = y_n] \, dF_{Y_1 \ldots Y_n}(y_1, \ldots, y_n)}{\mathsf{P}(Y_1 \in A_{1i_1}, \ldots, Y_n \in A_{ni_n})}. \tag{3}$$

Clearly, in order to compute (3), we need to know $\mathsf{E}[X \mid Y_1, \ldots, Y_n]$. If the entire joint distribution $F_{XY_1 \ldots Y_n}$ is not available, computation of h will not be possible in general. Another consideration in some applications is the computation of (3) in real time. If (3) is not computable in real time, all the different possible values of the right-hand side of (3) will have to be precomputed and stored. For an n-sensor system with N-component partitions, there are N^n different numbers to compute and store. Finally, if more sensors are added at a later date, there will be no way to take advantage of the work already done to develop the n-sensor system; all of the numbers given by (3) will have to be recomputed for an even larger value of n.

The preceding paragraph assumed that the partitions were given. If h is arbitrary and given, and the partitions $\{A_{ki}\}_{i=1}^{N}$ are given for $k \neq l$, then the remaining partition should satisfy (in order to minimize (2) [5])

$$y \in A_{li} \iff V_{li}(y) \leq V_{lj}(y) \qquad \text{for all } j = 1, \ldots, N. \tag{4}$$

where,

$$V_{li}(y) \triangleq \mathsf{E}[|\mathsf{E}[X \mid Y_1, \ldots, Y_n] - h_l(i-1)|^2 \mid Y_l = y],$$

and $h_l(i-1) \triangleq h(Z_1, \ldots, Z_{l-1}, i-1, Z_{l+1}, \ldots, Z_n)$. The approach in [5] was to use (3) and (4) as the basis of an algorithm for computing a locally optimal quantizer for a distributed estimation system. Briefly, one starts with an arbitrary initial quantizer and computes a function $h^{(1)}$ given by (3). Using $h^{(1)}$ and the initial partition, a new partition is generated using (4). One then starts over and uses the new partition to generate $h^{(2)}$ according to (3), and so on. The algorithm stops when the mean-square error given by (2) when $h = h^{(m)}$ (m is the iteration number) is small enough.

As the preceding discussion indicates, the computational size of this problem grows exponentially with the number of sensors n. Below we impose constraints on h so that the size of the problem of finding a locally optimal quantizer grows linearly with n.

III. Constraining the Fusion Center

Our approach [4] is to constrain the computational capabilities of the fusion center a priori as follows. We require that

$$\hat{X} = h(Z_1, \ldots, Z_n) = \sum_{k=1}^{n} g_k(Z_k), \tag{5}$$

where

$$g_k(Z_k) \triangleq \sum_{i=1}^{N} c_{ki} I_{\{i-1\}}(Z_k). \tag{6}$$

Remark: In spite of the sums in (5) and (6), the fusion center is performing a *non*linear operation on the input

data $Z_1, \ldots, Z_n$. In fact, since the Z_k are discrete random variables, the set of possible inputs does not constitute a vector space over $\mathbb{R}$. Similarly, each g_k in (6) is a nonlinear function of Z_k, and in (1), Z_k is a nonlinear function of Y_k.

Combining (5) and (6), and recalling that $Z_k = i - 1$ if and only if $Y_k \in A_{ki}$, we have

$$\hat{X} = \sum_{k=1}^{n} \sum_{i=1}^{N} c_{ki} I_{A_{ki}}(Y_k). \tag{7}$$

Clearly, $\hat{X}$ is a nonlinear function of $Y_1, \ldots, Y_n$. However, if the partitions at the sensors are fixed, choosing the $\{c_{ki}\}$ that minimize $\mathsf{E}[|X - \hat{X}|^2]$ is a linear-estimation problem whose solution is given by the usual normal equations. In this case, we will have Nn equations in Nn unknowns. Hence, the number of equations will grow linearly with the number of sensors n. The moments needed to write down the normal equations are

$$\mathsf{E}[I_{A_{ki}}(Y_k) I_{A_{lj}}(Y_l)] = \begin{cases} \mathsf{P}(Y_k \in A_{ki}) \delta_{ij}, & l = k, \\ \mathsf{P}(Y_k \in A_{ki}, Y_l \in A_{lj}), & l \neq k, \end{cases} \tag{8}$$

where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise, and

$$\mathsf{E}[X I_{A_{ki}}(Y_k)] = \int_{A_{ki}} \mathsf{E}[X \mid Y_k = y] \, dF_{Y_k}(y). \tag{9}$$

Note that one needs only the joint distributions of the form $F_{Y_k Y_l}$ and F_{XY_k} and not the entire joint distribution $F_{XY_1 \ \ldots \ Y_n}$.

Definition 1: Given a partition $\{A_{ki}\}_{i=1}^{N}$ for each sensor k, we write

$$\mathbf{A} = (\{A_{1i}\}_{i=1}^{N}, \ldots, \{A_{ni}\}_{i=1}^{N}).$$

We denote the procedure of solving the above-mentioned linear estimation problem by $\mathrm{SN}(\mathbf{A})$. Letting C denote the $n \times N$ matrix with elements c_{ki}, we write $C = \mathrm{SN}(\mathbf{A})$.

We now discuss how to find a good partition. If the matrix C is fixed, it is now very natural to ask how the best partition is characterized. To obtain the answer to this question, fix any $l = 1, \ldots, n$, and write

$$\mathsf{E}[|X - \hat{X}|^2] = \mathsf{E}[|X - g_l(Z_l) - \sum_{k \neq l} g_k(Z_k)|^2]. \tag{10}$$

The right-hand side of (10) can be expanded into nine terms; however, only five terms will involve Z_l. Denoting the sum of these five terms by J_l, we have

$$J_l = \mathsf{E}[g_l(Z_l)\{g_l(Z_l) - 2(X - \sum_{k \neq l} g_k(Z_k))\}].$$

Recalling that Z_l is a function of Y_l (cf. (1)), we can use the smoothing property of conditional expectation to write

$$J_l = \mathsf{E}[g_l(Z_l)\{g_l(Z_l) - 2(\mathsf{E}[X \mid Y_l] - \sum_{k \neq l} \mathsf{E}[g_k(Z_k) \mid Y_l])\}].$$

We can then write

$$J_l = \sum_{i=1}^{N} \int_{A_{li}} \varphi_{li}(y) \, dF_{Y_l}(y),$$

where $\varphi_{li}(y) \triangleq c_{li}(c_{li} - 2r_l(y))$ and

$$\begin{aligned} r_l(y) &\triangleq \mathsf{E}[X \mid Y_l = y] - \sum_{k \neq l} \mathsf{E}[g_k(Z_k) \mid Y_l = y] \\ &= \mathsf{E}[X \mid Y_l = y] \\ &\quad - \sum_{k \neq l} \sum_{j=1}^{N} c_{kj} \mathsf{P}(Y_k \in A_{kj} \mid Y_l = y). \end{aligned} \tag{11}$$

Clearly, if the $\{c_{ki}\}$ are fixed for all k and i, and if the partitions $\{A_{ki}\}_{i=1}^{N}$ are fixed for $k \neq l$, then we should put

$$y \in A_{li} \iff \varphi_{li}(y) \leq \varphi_{li'}(y) \quad \text{for all } i' = 1, \ldots, N.$$

If we assume that $c_{l1} < \cdots < c_{lN}$, then this is equivalent to

$$A_{li} = \left\{ y \in \mathbb{R} : \frac{c_{l,i-1} + c_{li}}{2} < r_l(y) \leq \frac{c_{li} + c_{l,i+1}}{2} \right\}. \tag{12}$$

(The choice of $<$ and $\leq$ is arbitrary and is made so that the $\{A_{li}\}_{i=1}^{N}$ will be disjoint.) Observe that the function r_l depends on the $\{c_{kj}\}_{j=1}^{N}$ for all $k \neq l$. Also, the set A_{li} in (12) is *not* an interval, but rather the inverse image of an interval. It is also important to observe that to compute r_l for $l = 1, \ldots, n$ only requires knowing the two-dimensional joint distributions F_{XY_l} and $F_{Y_k Y_l}$ for all k and l. An important consequence of this fact is that if we decide to add another sensor to measure, say Y_{n+1}, our prior knowledge of F_{XY_l} and $F_{Y_k Y_l}$ for $k, l \leq n$ can be reused. Of course, we would still need to obtain $F_{XY_{n+1}}$ and $F_{Y_k Y_{n+1}}$ for $k = 1, \ldots, n$.

We conclude this section with a final definition and a remark.

Definition 2: We introduce the procedure $U_l(C, \mathbf{A})$. Recall our notation in Definition 1. Let $\hat{\mathbf{A}} = U_l(C, \mathbf{A})$ be obtained from $\mathbf{A}$ by replacing $\{A_{li}\}_{i=1}^{N}$ with $\{\hat{A}_{li}\}_{i=1}^{N}$, where each $\hat{A}_{li}$ is given by the right-hand side of (12).

Remark: If $n = 1$ and $X = Y_1$, then the normal equations reduce to

$$\mathsf{P}(Y_1 \in A_{1i}) c_{1i} = \mathsf{E}[Y_1 I_{A_{1i}}(Y_1)],$$

or

$$c_{1i} = \frac{\int_{A_{1i}} y\, dF_{Y_1}(y)}{\mathsf{P}(Y_1 \in A_{1i})}.$$

Further, $r_1(y) = y$, and so

$$A_{1i} = \left\{ y \in \mathbb{R} : \frac{c_{1,i-1} + c_{1i}}{2} < y \leq \frac{c_{1i} + c_{1,i+1}}{2} \right\}.$$

In other words, we recover the classical Lloyd–Max conditions for locally optimal quantizers [6, 7].

IV. The Design Algorithm

Using the basic procedures SN and U_l, $l = 1, \ldots, n$, defined in the preceding section, there are two, almost identical, algorithms for generating approximately locally optimal quantizers for distributed estimation systems.

Algorithm 1

Let $\mathbf{A} = (\{A_{1i}\}_{i=1}^N, \ldots, \{A_{ni}\}_{i=1}^N)$ be given.
$C := \mathrm{SN}(\mathbf{A})$
loop1: **for** $l = 1$ **to** n
$\mathbf{A} := U_l(C, \mathbf{A})$
next l
$C := \mathrm{SN}(\mathbf{A})$
if stopping criterion not met, **go to** loop1
end

Algorithm 2

Let $\mathbf{A} = (\{A_{1i}\}_{i=1}^N, \ldots, \{A_{ni}\}_{i=1}^N)$ be given.
$C := \mathrm{SN}(\mathbf{A})$
loop2: **for** $l = 1$ **to** n
$\mathbf{A} := U_l(C, \mathbf{A})$
$C := \mathrm{SN}(\mathbf{A})$
next l
if stopping criterion not met, **go to** loop2
end

While the preceding algorithms appear simple enough, their implementation is nontrivial. The two main difficulties in implementing the algorithms are the computation of the function $r_l(y)$ in (11) and the characterization of the inverse images in (12). Note that even if $X, Y_1, \ldots, Y_n$ are jointly Gaussian, we cannot write (11) in closed form even if the A_{kj} are intervals. Hence, the sets A_{li} in (12) must be determined numerically and then a description of them must be stored in a suitable data structure.

A set of programs has been developed [8] to implement Algorithms 1 and 2 when provided with subroutines to compute the particular moments and probabilities for a given situation. Several examples of the form

$$Y_k = X + W_k, \quad k = 1, 2,$$

where X, W_1, and W_2 are statistically independent were considered.

Example [8, Example 8]: Let X have density

$$p(x) = \begin{cases} \frac{d}{b}\left[\frac{5}{4} - \cos\left(\frac{3\pi x}{2b}\right)\right], & |x| \leq b, \\ 0, & \text{otherwise}, \end{cases} \tag{13}$$

where $d \approx 0.3419$ is a normalization constant and $b = 2$ (see Fig. 2). We let W_1 and W_2 have the same density except that $b = 1$.

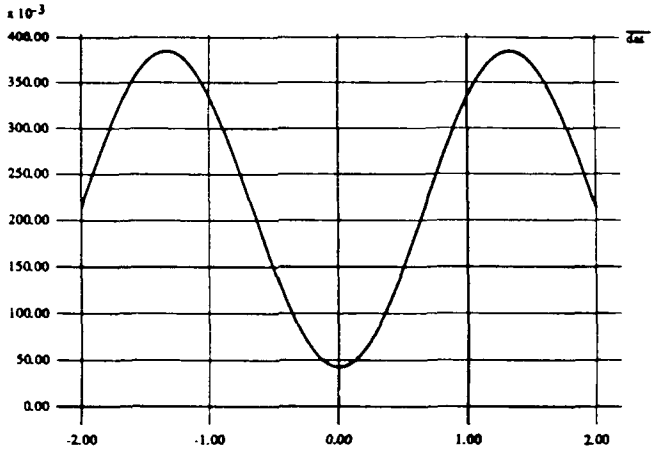


Fig. 2. Density $p(\cdot)$ in (13) with $b = 2$.

With $N = 8$ (3-bit quantizers), the Lloyd–Max partitions for the random variables Y_k, $k = 1, 2$, are identical and are given by

$$\begin{aligned} A_{k1} &= (-\infty, -1.3273] \\ A_{k2} &= (-1.3273, -1.2419] \\ A_{k3} &= (-1.2419, -1.0374] \\ A_{k4} &= (-1.0374, 0] \\ A_{k5} &= (0, 1.0374] \\ A_{k6} &= (1.0374, 1.2419] \\ A_{k7} &= (1.2419, 1.3273] \\ A_{k8} &= (1.3273, \infty). \end{aligned} \tag{14}$$

Solving the normal equations for $C = \mathrm{SN}(\mathbf{A})$, we have

$$\mathsf{E}[|X - \hat{X}|^2] = 0.18129.$$

Note that the minimum mean square error achievable by a linear estimator is 0.18534. Thus, just using the Lloyd–Max quantizers and doing linear estimation on Z_1, Z_2 can do better than pure linear estimation. Using Algorithm 1, we initialized $\mathbf{A}$ to the Lloyd–Max partition in (14). After 5 passes through the loop in Algorithm 1, the partitions

were

$$
\begin{aligned}
A_{11} &= (-\infty, -2.1802] \\
A_{12} &= (-2.8102, -1.7950] \cup (-1.3697, -0.8177] \\
A_{13} &= (-1.7950, -1.3697] \cup (-0.8177, -0.4109] \\
A_{14} &= (-0.4109, 0.0005686] \\
A_{15} &= (0.0005686, 0.4130] \\
A_{16} &= (0.4130, 0.8200] \cup (1.3739, 1.7979] \\
A_{17} &= (0.8200, 1.3739] \cup (1.7979, 2.1834] \\
A_{18} &= (2.1834, \infty)
\end{aligned}
$$

and

$$
\begin{aligned}
A_{21} &= (-\infty, -2.1056] \\
A_{22} &= (-2.1056, -0.6504] \\
A_{23} &= (-0.6504, -0.3207] \\
A_{24} &= (-0.3207, -0.1805] \\
A_{25} &= (-0.1805, 0.2863] \\
A_{26} &= (0.2863, 0.6240] \\
A_{27} &= (0.6240, 2.0960] \\
A_{28} &= (2.0960, \infty).
\end{aligned}
$$

The minimum mean square error for these partitions is

$$E[|X - \hat{X}|^2] = 0.12655,$$

which is more than a 30% improvement over the performance of the Lloyd–Max partition and over pure linear estimation.

Remarks: (*i*) After 5 passes through the algorithm, the mean square error was not significantly reduced.

(*ii*) The final partitions for the sensors are not the same, even though sensors 1 and 2 play interchangeable roles in this example. The reason for this is that the algorithm treats one sensor at a time.

(*iii*) As a general rule, it was found in [8] that Algorithm 2 yielded results almost identical to those of Algorithm 1.

V. Conclusion

We have developed an algorithm for the design of a distributed estimation system with n sensors and a single fusion center that is subject to communication and computation constraints. The algorithm uses only bivariate probability distributions and yields locally optimal estimators that satisfy the required system constraints.

While this work was motivated by problems in sensor fusion, the ideas can also be applied in a general nonlinear estimation context. In other words, estimators of the form (7) constitute a class of nonlinear estimators, and the algorithm presented here can be used to obtain a locally optimal nonlinear estimator from this class.

Acknowledgment

We thank Wael Wehbe for developing the programming system used to obtain the numerical results in Section IV.

References

[1] E. Ayanoglu, "On optimal quantization of noisy sources," *IEEE Trans. Inform. Theory*, vol. 36, no. 6, pp. 1450-1452, Nov. 1990.

[2] Y. Ephraim and R. M. Gray, "A unified approach for encoding clean and noisy sources by means of waveform and autoregressive model vector quantization," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 826-834, July 1988.

[3] T. J. Flynn and R. M. Gray, "Encoding of correlated observations," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 773-787, Nov. 1987.

[4] J. A. Gubner, "Constrained distributed estimation and quantization for distributed estimation systems," *Abstracts of Papers, 1990 IEEE International Symposium on Information Theory*, San Diego, CA, pp. 32-33, Jan. 1990.

[5] W. M. Lam and A. R. Reibman, "Quantizer design for decentralized estimation systems with communication constraints," *Proc. Twenty-Third Annual Conf. Inform. Sci. Syst.*, The Johns Hopkins University, Mar. 1989.

[6] S. P. Lloyd, "Least squares quantization in PCM," unpublished Bell Laboratories Memorandum, July 31, 1957; also *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 129-137, Mar. 1982.

[7] J. Max, "Quantizing for minimum distortion," *IRE Trans. Inform. Theory*, vol. IT-6, pp. 7-12, Mar. 1960.

[8] W. H. Wehbe, "Quantization for distributed estimation systems," M.S. report, Dep. Elect. Comp. Eng., Univ. of Wisconsin, Madison, 1992.

[9] Z. Zhang and T. Berger, "Estimation via compressed information," *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 198-211, Mar. 1988.

Appendix B

Reprint of Reference [8]

QUANTIZATION FOR DISTRIBUTED ESTIMATION SYSTEMS

by

WAEL HAIDAR WEHBE

A report submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

at the

UNIVERSITY OF WISCONSIN-MADISON

1992

1. Introduction

Distributed estimation systems lie at the heart of many important applications. For example, distributed estimation systems are employed by public power utilities to determine various loads and voltages both inside and outside their services areas. Improved estimation systems would permit them to provide electricity more economically. Another application of distributed systems is in the area of air travel control. In this case, many dispersed radars monitor aircraft in order to provide guidance and to avoid collisions. Better estimation systems would provide safer air travel.

In this report, we consider a distributed estimation system consisting of n sensors that transmit their findings to a common fusion center over a channel of positive, finite capacity. Using such a system, it is desired to estimate some quantity X, which cannot itself be observed. Because the communication channel has finite capacity, it is necessary for the sensors to quantize their measurements before transmission.

After some preliminary definitions in Section 2, we give a detailed overview of the constrained estimator approach in Section 3. In Section 4 we present an algorithm for quantization in distributed estimation systems (the multi-sensor case) for which the theory was outlined in Section 3. Section 4 also contains some numerical results obtained using this algorithm. Finally in Section 5 we give an overview of the code used to implement the algorithm presented in Section 4.

2. Background and Notation

Consider the distributed estimation system shown in Figure 1. This system consists of n sensor platforms that transmit their findings over a communication channel to a common fusion center. The sensors do not communicate with each other. Using this system some quantity, X, is to be estimated. However, only the sensor measurements $Y_1, \ldots, Y_n$, which are related to X, are available. Suppose that the findings, $Y_1, \ldots, Y_n$, of the n sensor platforms are transmitted to the fusion center over channels capable of transmitting at most R bits per second during a finite time interval $[0, T]$. Then each sensor k communicates the real number Y_k by sending RT-bit word to the fusion center. Hence, it is necessary for the sensors to

quantize their measurements.

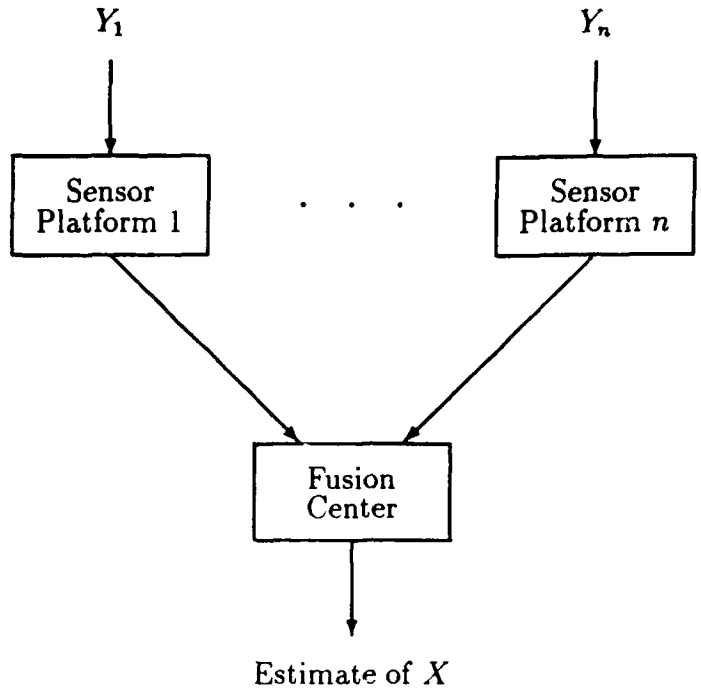


Figure 1: A Distributed Estimation System.

Let $X, Y_1, \ldots, Y_n$ be real-valued random variables on some probability space $(\Omega, \mathcal{F}, \mathsf{P})$. Since each measurement Y_k must be mapped into an RT-bit word, it is convenient to set $N = 2^{RT}$. We assume that each sensor k is equipped with a partition, $\{A_{ki}\}_{i=1}^{N}$, and that sensor k transmits the random variable

$$Z_k = \sum_{i=1}^{N} (i-1) I_{A_{ki}}(Y_k) \tag{1}$$

to the fusion center without error. (Here $I_A(y)$ denotes the indicator function of the set A; i.e., $I_A(y) = 1$ if $y \in A$ and $I_A(y) = 0$ otherwise.) Note that the event $\{Z_k = i-1\}$ is the same as the event $\{Y_k \in A_{ki}\}$.

Below we give an overview of the constrained estimator approach, which yields an algorithm for quantization in distributed estimation systems. We should note that the goal of quantization in such systems is to provide the best estimate of the unobservable X given the measurements $Y_1, \ldots, Y_n$ related to X. Our goal is not to reconstruct the measurements $Y_1, \ldots, Y_n$ at the fusion center.

3. The Constrained Estimator

Our approach is to constrain the computational capabilities of the fusion center a priori as follows. Denote the fusion center output by $\hat{X}$. We require that

$$\hat{X} = \sum_{k=1}^{n} g_k(Z_k),$$

where

$$g_k(Z_k) = \sum_{i=1}^{N} c_{ki} I_{\{i-1\}}(Z_k).$$

In other words, since $Z_k = i - 1$ if and only if $Y_k \in A_{ki}$,

$$\hat{X} = \sum_{k=1}^{n} \sum_{i=1}^{N} c_{ki} I_{A_{ki}}(Y_k). \tag{2}$$

Clearly $\hat{X}$ is a nonlinear function of Y_k. It is convenient to rewrite (2) in matrix notation. Let the matrices h and a be defined as:

$$h \triangleq [I_{A_{11}}(Y_1), I_{A_{12}}(Y_1), \ldots, I_{A_{n1}}(Y_n), \ldots, I_{A_{nN}}(Y_n)]^T$$
$$a \triangleq [c_{11}, c_{12}, \ldots, c_{n1}, \ldots, c_{nN}]^T.$$

Then the fusion center output becomes,

$$\hat{X} = a^T h. \tag{3}$$

When the partitions at the sensors are fixed, the problem of choosing a to minimize $\mathsf{E}[|X - \hat{X}|^2]$ is a simple linear estimation problem whose solution is given by the normal equations. That is, the optimal a is the solution of

$$Ra = r, \tag{4}$$

where $R \triangleq \mathsf{E}[hh^T]$ and $r \triangleq \mathsf{E}[Xh]$. To solve the normal equations, we have Nn equations in Nn unknowns. Hence, the number of equations will grow linearly with the number of sensors n. For example if $n = 4$ and $N = 2$, there will be 8 equations. The moments needed to write down the normal equations are

$$\mathsf{E}[I_{A_{ki}}(Y_k) I_{A_{lj}}(Y_l)] = \begin{cases} \mathsf{P}(Y_k \in A_{ki})\delta_{ij}, & \text{if } l = k, \\ \mathsf{P}(Y_k \in A_{ki}, Y_l \in A_{lj}), & \text{if } l \neq k, \end{cases} \tag{5}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise, and

$$\mathsf{E}[\,X I_{A_{ki}}(Y_k)\,] \;=\; \int_{A_{ki}} \mathsf{E}[\,X \mid Y_k = y\,]\, dF_{Y_k}(y). \tag{6}$$

Definition 1. Given a partition $\{A_{ki}\}_{i=1}^{N}$ for each sensor k, we write

$$\mathbf{A} \;=\; (\{A_{1i}\}_{i=1}^{N}, \ldots, \{A_{ni}\}_{i=1}^{N}).$$

We denote the procedure of solving the above-mentioned linear estimation problem by $\mathrm{SN}(\mathbf{A})$. Letting C denote the $n \times N$ matrix with elements c_{ki}, we write $C = \mathrm{SN}(\mathbf{A})$. Clearly C and a contain the same data, simply rearranged.

How to find a good partition. If the matrix C is fixed, it is now very natural to ask how the best partition is characterized. To obtain the answer to this question, fix any $l = 1, \ldots, n$, and write

$$\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; \mathsf{E}[\,|X - g_l(Z_l) - \sum_{k \neq l} g_k(Z_k)|^2\,]. \tag{7}$$

The right-hand side of (7) can be expanded into nine terms; however, only five terms will involve Z_l. Denoting the sum of those five terms by J_l, we have

$$J_l \;=\; \mathsf{E}[\,g_l(Z_l)\{g_l(Z_l) - 2(X - \sum_{k \neq l} g_k(Z_k))\}\,].$$

Recalling that Z_l is a function of Y_l (cf. (1)), we can use the smoothing property of conditional expectation to write,

$$J_l \;=\; \mathsf{E}[\,g_l(Z_l)\{g_l(Z_l) - 2(\mathsf{E}[\,X \mid Y_l\,] - \sum_{k \neq l} \mathsf{E}[\,g_k(Z_k) \mid Y_l\,])\}\,].$$

We can then write

$$J_l \;=\; \sum_{i=1}^{N} \int_{A_{li}} \underbrace{c_{li}\{c_{li} - 2(\mathsf{E}[\,X \mid Y_l = y\,] - \sum_{k \neq l} \mathsf{E}[\,g_k(Z_k) \mid Y_l = y\,])\}}_{\varphi_{li}(y)}\, dF_{Y_l}(y).$$

Clearly, if the $\{c_{ki}\}$ are fixed for all k and i, and if the partitions $\{A_{ki}\}_{i=1}^{N}$ are fixed for $k \neq l$, then we should put

$$y \in A_{li} \iff \varphi_{li}(y) \leq \varphi_{li'}(y) \qquad \text{for all } i' = 1, \ldots, N. \tag{8}$$

We can obtain a simpler expression for (8) by setting

$$\begin{aligned} r_l(y) &\triangleq \mathsf{E}[X \mid Y_l = y] - \sum_{k \neq l} \mathsf{E}[g_k(Z_k) \mid Y_l = y] \\ &= \mathsf{E}[X \mid Y_l = y] - \sum_{k \neq l} \sum_{j=1}^{N} c_{kj} \mathsf{P}(Y_k \in A_{kj} \mid Y_l = y). \end{aligned} \tag{9}$$

We can then write $\varphi_{li}(y) = c_{li}(c_{li} - 2r_l(y))$. If we assume that $c_{l1} < \cdots < c_{lN}$, then (8) is equivalent to

$$A_{li} = \left\{ y \in \mathbb{R} : \frac{c_{l,i-1} + c_{li}}{2} < r_l(y) \leq \frac{c_{li} + c_{l,i+1}}{2} \right\}. \tag{10}$$

Observe that the function r_l depends on the $\{c_{kj}\}_{j=1}^{N}$ for all $k \neq l$. Also, the set A_{li} in (10) is *not* an interval, but rather the inverse image of an interval.

Definition 2. Recall our notation in Definition 1. Let

$$U_l(C, \mathbf{A}) \triangleq (\{A_{1i}\}_{i=1}^{N}, \ldots, \{A_{l-1,i}\}_{i=1}^{N}, \{\hat{A}_{li}\}_{i=1}^{N}, \{A_{l+1,i}\}_{i=1}^{N}, \ldots, \{A_{ni}\}_{i=1}^{N}),$$

where each $\hat{A}_{li}$ is given by the right-hand side of (10).

Finally to evaluate the performance of the constrained estimator algorithm the mean square error is computed. If $\hat{X}$ satisfies (3) and a satisfies (4), the mean square error is:

$$\begin{aligned} \mathsf{E}[|X - \hat{X}|^2] &= \mathsf{E}[X^2 + \hat{X}^2 - 2X\hat{X}] \\ &= \mathsf{E}[X^2 + (a^T h)^2 - 2a^T X h] \\ &= \mathsf{E}[X^2] - 2a^T r + a^T R a \\ &= \mathsf{E}[X^2] - a^T r \\ &= \mathsf{E}[X^2] - \sum_{k=1}^{n} \sum_{i=1}^{N} c_{ki} \mathsf{E}[X I_{A_{ki}}(Y_k)]. \end{aligned} \tag{11}$$

Remark. If $n = 1$ and $X = Y_1$, then the normal equations reduce to

$$\mathsf{P}(Y_1 \in A_{1i}) c_{1i} = \mathsf{E}[Y_1 I_{A_{1i}}(Y_1)],$$

or

$$c_{1i} = \frac{\int_{A_{1i}} y \, dF_{Y_1}(y)}{\mathsf{P}(Y_1 \in A_{1i})}.$$

Further, $r_1(y) = y$, and so

$$A_{1i} = \left\{ y \in \mathbb{R} : \frac{c_{1,i-1} + c_{1i}}{2} < y \leq \frac{c_{1i} + c_{1,i+1}}{2} \right\}.$$

In other words, we recover the classical Lloyd–Max conditions for locally optimal quantizers [1, 2].

4. Numerical Results

Using the basic procedures SN and U_l, $l = 1, \ldots, n$, defined in the preceding section, there is an obvious algorithm for obtaining (approximate) locally optimal quantizers for distributed estimation systems.

Algorithm 1

```
        Let A = ({A_1i}_{i=1}^N, ..., {A_ni}_{i=1}^N) be given.
        C := SN(A)
loop1:  for l = 1 to n
            A := U_l(C, A)
        next l
        C := SN(A)
        if stopping criterion not met, go to loop1
        end
```

While the preceding algorithm appears simple enough, its implementation is not trivial. The two main difficulties in implementing the algorithm are the computation of the function $r_l(y)$ in (9) and the characterization of the inverse images in (10). Note that even if $X, Y_1, \ldots, Y_n$ are jointly Gaussian, we cannot write (9) in closed form even if the A_{kj} are intervals. Hence, the sets A_{li} in (10) must be determined numerically and then a description of them must be stored in some data structure.

Algorithm 1 has been programmed, and we have run several numerical examples. In all the examples below, we took

$$Y_k = X + W_k, \qquad k = 1, \ldots, n, \tag{12}$$

where $X, W_1, \ldots, W_n$ are statistically independent.

Example 1. We first took $n = 1$ sensor, $N = 4$ (2-bit quantizers), and $Y_1 = X \sim N(0,1)$; i.e, X is normal with mean 0 and variance 1. In other words, we used our algorithm to solve the classical Lloyd–Max problem. From several initial partitions, our algorithm quickly converged to the optimal quantizer,

$$\begin{aligned} A_{k1} &= (-\infty, -0.9816] \\ A_{k2} &= (-0.9816, 0] \\ A_{k3} &= (0, 0.9816] \\ A_{k4} &= (0.9816, \infty), \end{aligned}$$

which was originally reported in [2, Table I].

Example 2. Suppose $n = 2$ sensors, $N = 4$ (2-bit quantizers), where $X \sim N(0, 0.9)$ and $W_1, W_2 \sim N(0, 0.1)$. Since Y_1 and Y_2 are $N(0,1)$ we take the initial partition for each sensor to be the Lloyd–Max partition found in Example 1. Taking $C = \mathrm{SN}(\mathbf{A})$, we find that

$$\mathsf{E}[|X - \hat{X}|^2] = 0.113.$$

After 6 passes through the loop in Algorithm 1, we find that

$$\begin{aligned} A_{11} &= (-\infty, -1.28] \\ A_{12} &= (-1.28, 0] \\ A_{13} &= (0, 1.28] \\ A_{14} &= (1.28, \infty) \end{aligned} \qquad \text{and} \qquad \begin{aligned} A_{21} &= (-\infty, -0.843] \\ A_{22} &= (-0.843, 0] \\ A_{23} &= (0, 0.843] \\ A_{24} &= (0.843, \infty) \end{aligned}$$

and

$$\mathsf{E}[|X - \hat{X}|^2] = 0.107,$$

which is about a 5% reduction in the mean square error.

Example 3. Let $n = 2$ sensors, $N = 4$ (2-bit quantizers), where this time $X \sim U(-0.9, 0.9)$; i.e., uniform over the interval $(-0.9, 0.9)$. We take $W_1, W_2 \sim U(-0.11, 0.11)$. Clearly, Y_k, $k = 1, 2$, has a density given by the convolution of two uniform densities. We initialize the

partition of each sensor to its Lloyd–Max partition, which in this case is ($k = 1, 2$)

$$\begin{aligned} A_{k1} &= (-\infty, -0.4546] \\ A_{k2} &= (-0.4546, 0] \\ A_{k3} &= (0, 0.4546] \\ A_{k4} &= (0.4546, \infty). \end{aligned}$$

Taking $C = \mathrm{SN}(\mathbf{A})$, we find that

$$\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; 0.01358.$$

After 21 passes through the loop of Algorithm 1, we find that

$$\begin{aligned} A_{11} &= (-\infty, -0.6390] \\ A_{12} &= (-0.6390, -0.1062] \\ A_{13} &= (-0.1062, 0.6224] \\ A_{14} &= (0.6224, \infty) \end{aligned} \qquad \text{and} \qquad \begin{aligned} A_{21} &= (-\infty, -0.3929] \\ A_{22} &= (-0.3929, -0.1450] \\ A_{23} &= (-0.1450, 0.3620] \\ A_{24} &= (0.3620, \infty) \end{aligned}$$

and

$$\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; 0.00895,$$

which is a reduction of about 34.5% in the mean square error. We should note that after only 8 passes the mean square error had been reduced by about 33%.

Example 4. We modify the previous example by increasing the partition size from $N = 4$ (2-bit quantizers) to $N = 64$ (8-bit quantizers). We initialize the partition of each sensor to its Lloyd–Max partition. Taking $C = \mathrm{SN}(\mathbf{A})$, we find that the initial mean square error is,

$$\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; 0.00197.$$

Algorithm 1 did not improve substantially the initial mean square error; however, the value 0.00197 is smaller than the minimum mean square error obtainable by linear estimation, which is 0.0020012.

Example 5. Suppose $n = 2$ sensors, $N = 4$ (2-bit quantizers), where $X \sim U(-0.9, 0.9)$, $W_1 \sim U(-0.11, 0.11)$ and the density for W_2 is

$$f_{W_2}(w) \;=\; \begin{cases} \dfrac{a}{2\tan^{-1}(ba)(1 + (aw)^2)}, & |\,w\,| \le b, \\ \\ 0, & \text{otherwise}, \end{cases} \tag{13}$$

where $b = 0.11$ and $a = 1$. We initialize the partition of each sensor to its Lloyd-Max partition. We have

$$\begin{aligned} A_{11} &= (-\infty, -0.4546] \\ A_{12} &= (-0.4546, 0] \\ A_{13} &= (0, 0.4546] \\ A_{14} &= (0.4546, \infty) \end{aligned} \quad \text{and} \quad \begin{aligned} A_{21} &= (-\infty, -0.4456] \\ A_{22} &= (-0.4456, 0] \\ A_{23} &= (0, 0.4456] \\ A_{24} &= (0.4456, \infty). \end{aligned}$$

Taking $C = \mathrm{SN}(\mathbf{A})$, we find that

$$\mathsf{E}[|X - \hat{X}|^2] = 0.0136.$$

After 21 passes through the loop in Algorithm 1, we find that

$$\begin{aligned} A_{11} &= (-\infty, -0.6390] \\ A_{12} &= (-0.6390, 0.1066] \\ A_{13} &= (0.1062, 0.6226] \\ A_{14} &= (0.6226, \infty) \end{aligned} \quad \text{and} \quad \begin{aligned} A_{21} &= (-\infty, -0.3927] \\ A_{22} &= (-0.3927, -0.1447] \\ A_{23} &= (-0.1447, 0.3624] \\ A_{24} &= (0.3624, \infty) \end{aligned}$$

and

$$\mathsf{E}[|X - \hat{X}|^2] = 0.00894,$$

which is about a 34.3% reduction in the mean square error. We should note that after only 8 passes the mean square error was reduced by 33.5%. These results are almost identical to those in Example 3; this is not surprising, since taking $a = 1$ in (13) implies that f_{W_2} is nearly flat, and hence W_2 is nearly uniform as in Example 3.

Example 6. Same as Example 5, except we replace $a = 1$ by $a = 32$. We initialize the partition of each sensor to the Lloyd-Max partition, which is

$$\begin{aligned} A_{11} &= (-\infty, -0.4546] \\ A_{12} &= (-0.4546, 0] \\ A_{13} &= (0, 0.4546] \\ A_{14} &= (0.4546, \infty) \end{aligned} \quad \text{and} \quad \begin{aligned} A_{21} &= (-\infty, -0.4481] \\ A_{22} &= (-0.4481, 0] \\ A_{23} &= (0, 0.4481] \\ A_{24} &= (0.4481, \infty). \end{aligned}$$

Taking $C = \mathrm{SN}(\mathbf{A})$, we find that

$$\mathsf{E}[|X - \hat{X}|^2] = 0.0136.$$

After 10 passes through the loop in Algorithm 1, we find that

$$\begin{aligned} A_{11} &= (-\infty, -0.6374] \\ A_{12} &= (-0.6374, 0.1462] \\ A_{13} &= (0.1462, 0.6160] \\ A_{14} &= (0.6160, \infty) \end{aligned} \quad \text{and} \quad \begin{aligned} A_{21} &= (-\infty, -0.3852] \\ A_{22} &= (-0.3852, 0.0913] \\ A_{23} &= (0.0913, 0.3496] \\ A_{24} &= (0.3496, \infty), \end{aligned}$$

and

$$\mathsf{E}[|X - \hat{X}|^2] = 0.00797,$$

which is about a 41.4% reduction in the mean square error. We should note that after only 2 passes the mean square error was reduced by 37.2%.

Example 7. Same as Example 5, but we change the density of W_2 to

$$f_{W_2}(w) = \begin{cases} \dfrac{0.3419}{b}\left[\dfrac{5}{4} - \cos\left(\dfrac{3\pi w}{2b}\right)\right], & |w| \le b, \\ 0, & \text{otherwise}, \end{cases} \tag{14}$$

where $b = 0.11$. We initialize the partition of each sensor to the Lloyd-Max partition; i.e.,

$$\begin{aligned} A_{11} &= (-\infty, -0.4546] \\ A_{12} &= (-0.4546, 0] \\ A_{13} &= (0, 4546] \\ A_{14} &= (0.4546, \infty) \end{aligned} \quad \text{and} \quad \begin{aligned} A_{21} &= (-\infty, -0.4445] \\ A_{22} &= (-0.4445, 0] \\ A_{23} &= (0, 0.4445] \\ A_{24} &= (0.4445, \infty). \end{aligned}$$

Taking $C = \mathrm{SN}(\mathbf{A})$, we find that

$$\mathsf{E}[|X - \hat{X}|^2] = 0.01358.$$

After 15 passes through the loop in Algorithm 1, we find that

$$\begin{aligned} A_{11} &= (-\infty, -0.6156] \\ A_{12} &= (-0.6156, -0.1088] \\ A_{13} &= (-0.1088, 0.6234] \\ A_{14} &= (0.6234, \infty) \end{aligned} \quad \text{and} \quad \begin{aligned} A_{21} &= (-\infty, -0.3570] \\ A_{22} &= (-0.3570, 0.1328] \\ A_{23} &= (0.1328, 0.3721] \\ A_{24} &= (0.3721, \infty), \end{aligned}$$

and

$$\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; 0.00938,$$

which is about a 31% reduction in the mean square error. We should note that after 3 passes the mean square error was reduced by 25.5%.

Example 8. Let $n = 2$ sensors, $N = 8$ (3-bit quantizers), where this time X has the density function shown in (14) where $b = 2$. We take W_1, W_2 to have the same density function of X, except we replace $b = 2$ by $b = 1$. We initialize the partition of each sensor to its Lloyd–Max partition, which in this case is $(k = 1, 2)$

$$\begin{aligned}
A_{k1} &= (-\infty, -1.3273] \\
A_{k2} &= (-1.3273, -1.2419] \\
A_{k3} &= (-1.2419, -1.0374] \\
A_{k4} &= (-1.0374, 0] \\
A_{k5} &= (0, 1.0374] \\
A_{k6} &= (1.0374, 1.2419] \\
A_{k7} &= (1.2419, 1.3273] \\
A_{k8} &= (1.3273, \infty).
\end{aligned}$$

Taking $C = \mathrm{SN}(\mathbf{A})$, we find that

$$\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; 0.18129.$$

After only 5 passes through the loop of Algorithm 1, we find that

$$\begin{aligned}
A_{11} &= (-\infty, -2.1802] \\
A_{12} &= (-2.8102, -1.7950] \cup (-1.3697, -0.8177] \\
A_{13} &= (-1.7950, -1.3697] \cup (-0.8177, -0.4109] \\
A_{14} &= (-0.4109, 0.0005686] \\
A_{15} &= (0.0005686, 0.4130] \\
A_{16} &= (0.4130, 0.8200] \cup (1.3739, 1.7979] \\
A_{17} &= (0.8200, 1.3739] \cup (1.7979, 2.1834] \\
A_{18} &= (2.1834, \infty)
\end{aligned}$$

$$
\begin{aligned}
A_{21} &= (-\infty, -2.1056] \\
A_{22} &= (-2.1056, -0.6504] \\
A_{23} &= (-0.6504, -0.3207] \\
A_{24} &= (-0.3207, -0.1805] \\
A_{25} &= (-0.1805, 0.2863] \\
A_{26} &= (0.2863, 0.6240] \\
A_{27} &= (0.6240, 2.0960] \\
A_{28} &= (2.0960, \infty)
\end{aligned}
$$

and

$$
\mathsf{E}[\,|X - \hat{X}|^2\,] \;=\; 0.12655,
$$

which is a reduction of about 30.1% in the mean square error. We should also point that the value 0.12655 is a 31.7% reduction of the minimum mean square error obtainable by linear estimation, which is 0.18534.

5. Conclusion

The best estimate of some quantity X, in a distributed estimation system, given some measurements say $Y_1, \ldots, Y_n$, is the conditional expectation of X given $Y_1, \ldots, Y_n$, denoted $\mathsf{E}[\,X|Y_1, \ldots, Y_n\,]$. However, the functional form of $\mathsf{E}[\,X|Y_1, \ldots, Y_n\,]$ as a function of $Y_1, \ldots, Y_n$ is often difficult, if not impossible to determine and it is not possible to transmit real-valued quantities without distortion. In this report we solved those problems by showing the feasibility and efficiency of an algorithm that was specifically directed to address the distributed nature of the system. In Section 3 we presented the constrained estimator approach, which is an algorithm for quantization for distributed estimation systems. The numerical results in Section 4 indicated that in general the constrained estimator performs better than the Lloyd-Max estimator when N is small ($N = 4$, 2-bit quantizers) as shown in Examples 2,3, and 5–7, however when N is large ($N = 64$, 8-bit quantizers) the constrained estimator does not improve the results of the Lloyd-Max estimator significantly as shown in Example 4.

An other conclusion can be drawn from the numerical results. The constrained estimator is a nonlinear estimator as pointed out in Section 2. This is shown in Example 4 and more

clearly in Example 8. In fact, the nonlinear constrained estimator performs better or in the worst case as well as the linear estimation provided that N is chosen appropriately.

Appendix A. An Overview of the Code

In this section we discuss in detail the code that was used to implement the constrained estimator algorithm presented in Section 4. But first we turn our attention to the few changes that we made to the equations presented in Sections 3 and 4. In view of the structure described in (12), it is convenient to express the quantities in (5), (6), and (9) in terms of the univariate densities $f_X(x)$ and $f_{W_i}(w)$. We find that

$$\mathsf{P}(Y_k \in A) \;=\; \int_{-\infty}^{\infty} \mathsf{P}(W_k \in A - x)\, f_X(x)\, dx, \tag{15}$$

$$\mathsf{P}(Y_k \in A, Y_l \in B) \;=\; \int_{-\infty}^{\infty} \mathsf{P}(W_k \in A - x)\, \mathsf{P}(W_l \in B - x)\, f_X(x)\, dx, \quad k \neq l, \tag{16}$$

$$\mathsf{E}[\, X I_A(Y_k)\,] \;=\; \int_{-\infty}^{\infty} \mathsf{P}(W_k \in A - x)\, f_X(x)\, dx, \tag{17}$$

$$\mathsf{P}(Y_k \in A \mid Y_l = y) \;=\; \frac{\displaystyle\int_{-\infty}^{\infty} \mathsf{P}(W_k \in A - x)\, f_{W_l}(y - x)\, f_X(x)\, dx}{f_{Y_l}(y)}, \quad k \neq l, \tag{18}$$

$$\mathsf{E}[\, X \mid Y_l = y\,] \;=\; \frac{\displaystyle\int_{-\infty}^{\infty} x\, f_{W_l}(y - x)\, f_X(x)\, dx}{f_{Y_l}(y)}, \tag{19}$$

where, of course,

$$f_{Y_l}(y) \;=\; \int_{-\infty}^{\infty} f_{W_l}(y - x)\, f_X(x)\, dx, \tag{20}$$

and

$$\mathsf{P}(W_k \in A - x) \;=\; \int_{A-x} f_{W_k}(w)\, dw.$$

In particular, if $A = (a, b]$, $A - x = (a - x, b - x]$, and

$$\mathsf{P}(W_k \in A - x) \;=\; \int_{a-x}^{b-x} f_{W_k}(w)\, dw. \tag{21}$$

Now that we mentioned the few changes needed to make the implementation of our algorithm easier, we are ready to discuss the code used in detail. As shown in Algorithm 1,

the first element that is needed to implement this algorithm is an arbitrary initial partition **A**. The partition **A** is defined as:

$$\mathbf{A} = (\{A_{1i}\}_{i=1}^{N}, \ldots, \{A_{ni}\}_{i=1}^{N})$$

where $\{A_{ki}\}_{i=1}^{N}$ is a partition for sensor k. We call A_{ki} the ith *subpartition* of sensor k.

We now discuss the procedure that read and stored the initial partition **A**. The initial partition is stored in the file `start`. The lower and upper limits of the intervals of each subpartition 1 through N are listed in succession and are separated by the pair $-1, -1$. This is repeated for each sensor 1 through n. We should note that the interval $(-\infty, \infty)$ is stored as the pair $0, -1$ while the interval $(-\infty, a)$ is stored as the pair $a+2, a$, and (a, ∞) is stored as the pair $a, a-3$.

For example, if $n = 1$ and $N = 2$ then the partition **A** is defined as $\mathbf{A} = (\{A_{1i}\}_{i=1}^{2})$, and if $A_{11} = (-\infty, 3] \cup (4, \infty)$ and $A_{12} = (3, 4]$, then the file `start` will contain the following lines:

```
 5,   3
 4,   1
-1,  -1
 3,   4
-1,  -1
```

This listing, available in file `start` is read and then stored in a 3-dimensional array `AR(I,J,K)` by the routine `ReadAR`. The first index `I` of the array `AR(I,J,K)` indicates which of the sensors from 1 through n we are dealing with, the second index `J` (`J=1,...,N`) indicates which subpartition we are pointing to, and finally, the third index `K` indicates if the number stored is the upper or lower endpoint of an interval of a subpartition at `J` of sensor `I`. For example, the set A_{11} described above in the file `start` will be stored in `AR` as follows:

$$\begin{aligned} \texttt{AR}(1,1,1) &= 5 \\ \texttt{AR}(1,1,2) &= 3 \end{aligned} \quad \text{and} \quad \begin{aligned} \texttt{AR}(1,1,3) &= 4 \\ \texttt{AR}(1,1,4) &= 1, \end{aligned}$$

and A_{12} will be stored in `AR` as follows:

$$\begin{aligned} \texttt{AR}(1,2,1) &= 3 \\ \texttt{AR}(1,2,2) &= 4. \end{aligned}$$

The routine **ReadAR** reads the file **start** and returns the array **AR** containing the initial partition. The key for the proper operation of the routine is the knowledge of the number of sensors, n, denoted by **NS** in the programs, and knowledge of the number of partitions, N denoted by **N** in the programs. The routine reads every pair of elements from **start** and stores them in **AR(I,J,K)** and **AR(I,J,K+1)**, this process is repeated until a pair of -1s occurs indicating the subpartition is updated and **J** becomes **J+1** and the next subpartition is read. When all subpartitions of intervals of a sensor **I** are read (**J=N**) and stored in **AR**, the index indicating the number of sensors is updated, and **I** becomes **I+1**. The whole process is then repeated until **I=NS**; at this point the partition **A** has been stored in **AR**.

Now that the initial partition **A** is read and stored in the matrix **AR(I,J,K)**, the next step is to compute the initial elements $\{c_{ki}\}$. To do that we generate the two matrices R and r as defined in (4). In the programs, R is called **R** and r is called **Mean**.

The routine **GenR(R)** returns the matrix **R** defined in (4) using the routine **SumP(L,J,K,I)** that generates the moments shown in (5), (15), and (16). Two nested do loops were used; the first loop generated all the elements of **R** and stored them in the $N^2n^2 \times 1$ matrix **Temp2**, the second do loop stored appropriately the elements of **Temp2** in the $Nn \times Nn$ matrix **R**.

The routine **GenM(Mean)** returns the matrix r denoted as **Mean** using the routine **SumM (K,I)** that generates the moments shown in (17). In this case one nested loop is used, this nested do loop will generate the $Nn \times 1$ matrix **Mean**.

Now that the matrices **R** and **Mean** are available, the routine **Solve(R,Mean,C,MSE)** computes the coefficients $\{c_{ki}\}$ stored in the $n \times N$ matrix **C**, where each row indicates a different sensor and the elements of a given row k are the coefficients of the sensor k that minimizes $\mathsf{E}[|X - \hat{X}|^2]$. This routine also returns the mean square error **MSE**, which is computed using (11). Since the normal equations used for obtaining the $\{c_{ki}\}$ are ill conditioned, we employed the singular-value decomposition (SVD) as discussed in [3]. To obtain the SVD of **R** we used the NAG routine **F02WEF**.

Finally, the routine **SN(AR,C)** will return the $n \times N$ matrix **C** containing the different elements $\{c_{ki}\}$ obtained by solving the mentioned linear estimation problem defined in (4) given the initial partition **A** stored in the array **AR**. This routine is used in the main program and is equivalent to the procedure SN in Definition 1.

We have just shown how to find the elements $\{c_{ki}\}$ that will minimize $\mathsf{E}[\,|X - X|^2\,]$ given a partition **A** using the procedure `SN(AR,C)`. The next step is to present the procedures that we used to improve a partition **A**. As we seek to improve a partition **A** given a matrix `C`, we have a major hurdle to overcome. The first part of that hurdle is the computation of the function $r_l(y)$ in (9), the second part is to characterize the inverse images in (10).

The function $r_l(y)$ is obtained by `RofL(Y)`; `RofL(Y)` calls two different routines, `EC(Ind,Y)` and `CP(Y)`. The function `EC(Ind,Y)` will compute $\mathsf{E}[\,X \mid Y_l = y\,]$ shown in (19). The second part of (9), which depends on the $\{c_{ki}\}$, is computed by `CP(Y)`. To compute $\mathsf{P}(Y_k \in A_{kj} \mid Y_l = y)$ defined in (18), `SumcP(K,J,Y)` is used. Since a given subpartition A_{kj} could be a union of intervals, `SumcP` computes the conditional probability at each interval using `Fun4(LL,UL,Y)` and adds the results together.

Before explaining how the set defined in (10) was obtained to best characterize the partition **A**, we discuss two routines that were used in this process, **`Levels`** and **`Root`**. The routine `Levels` computes the different levels needed to generate the partition $\{A_{ki}\}_{i=1}^{N}$ for a given sensor k. If there are N subpartitions we will have $N-1$ levels all stored in an array `Lev(I)` defined as:

$$\texttt{Lev(I)} \;=\; \frac{\texttt{C(sensorindex, I)} + \texttt{C(sensorindex, I+1)}}{2}$$

where $I = 1, \ldots, N-1$.

Given the function `RofL(Y)` and an interval `[LL,UL]` the routine **`Root`** will find a simple root in this interval. If more than one root exists in a given interval, an error message is issued. To compute the roots the NAG routine `C05ADF` is used.

To obtain a new partition **A**, two main routines are used: **`ListofZero`** and **`Newpart`**. **`ListofZero`** will find all the zeros of `RofL(Y)` − `Lev(I)`. To do this, we sample $r_l(y)$ for **`Numpt`** values of y. The values of y are stored in the array `Yp`. The values of $r_l(y)$ are stored in the array `Rp`. The routine **`ListofZero`** will search for a change of sign of **`Rp(J)-Lev(I)`**. A change of sign is assumed to indicate that at most one root exists in the interval `Y(J)` and `Y(J+1)`. This assumes that enough samples of $r_l(y)$ are taken. The zeros are stored in the array `Zero(I,K)`, where the index `I` indicates to which level the zero corresponds. In addition, the nearest value to the left and to the right of $r_l(y)$ are stored respectively

in **Testpt(I,K,1)** and **Testpt(I,K,2)**. The number of zeros for each level is stored in the array **NumofZero**.

The routine **Newpart** will use the list of the zeros of the different levels and the test points to obtain the new partition **A**. The test points will be used to take a peak at the left or right of a zero and will enable us to determine if the range of $r_l(y)$ in an interval formed by two consecutive zeros lies above or below a certain level or between two levels. To determine the intervals of y for which $r_l(y)$ is between two levels, we will first merge in order using **MergeSort** the zeros and their corresponding test points of the two levels in the arrays **Mzero** and **Mtestpoint** respectively and then find the desired intervals.

Finally the routine **UL(k,AR)** where $k = 1, \ldots, n$ will return a new partition of $\{A_{ki}\}_{i=1}^{N}$ denoted by $\{\hat{A}_{ki}\}_{i=1}^{N}$ and stored in the array **AR** given the $N \times n$ array **C** that contains the different elements c_{ki}. This routine is used in the main program. It plays the role of $U_l(C, \mathbf{A})$ defined in Definition 2 and used in Algorithm 1.

To study the performance of Algorithm 1 we computed the mean square error $\mathsf{E}[\,|X - \hat{X}|^2\,]$ defined in (11) each time the partition **A** and its corresponding coefficients c_{ki} are updated. The latest mean square error is compared with the previous value. In case of any substantial improvement another iteration is taken, otherwise we stop running the program.

Appendix A. A Listing of the Code

```
c                define.f

c This variable stores the Mean Square of the distribution
c of X U~(-A,A).
  DOUBLE PRECISION VARX
  COMMON/BLOCK45/VARX



c NS ---> number of sensors
c N  ---> number of partitions
c MAX=2*SUBP, where SUBP is the number of subpartitions
c The data of the variables NS,N,SUBP are read
c respectivelly from the file init.dat.
  INTEGER NS,N,MAX
  COMMON/BLOCK5/NS,N,MAX


c A1 =NS*N    and    B1=NS*NS*N*N
  INTEGER A1,B1
  COMMON/BLOCK6/A1,B1



c COND ---> indicates if the conditional probabilities are
c           conditioned on Sensor1 or Sensor2 (Z1 or Z2).
c SNUMB---> indicates the SENSOR NUMBER we are dealing with
c           If we want the LLOYD-MAX partition for SENSOR 2
c           then SNUMB is 2.
c           If we are running the algorithm then SNUMB=NS.
  INTEGER COND,SNUMB
  COMMON/BLOCK7/COND
  COMMON/BLOCK25/SNUMB



c NNS ---> MAXIMUM number of sensors
c NN  ---> MAXIMUM number of partitions
c NSUBP---> MAXIMUM number of subpartitions
  INTEGER NNS,NN,NSUBP,NMAX,AS,NA
  PARAMETER (NNS=2,NN=64,NSUBP=30,NMAX=2*NSUBP,AS=NNS*NN,NA=AS*AS)



c The variable LEVEL is needed to find all the values of
c y for which the equation LEVEL=RofL(y) holds.
  DOUBLE PRECISION LEVEL
  COMMON/BLOCK23/LEVEL



c LMAX -->indicates if we are computing the LLOYD-MAX partition
c        or not,LMAX =.TRUE. if we are ,LMAX=.false if we are not
  LOGICAL LMAX
  COMMON/BLOCK24/LMAX


c Rp is an array that contains the values of RofL at specific
c values of y.Yp is an array that contains the corresponding
c values of y for Rp.
c The MAXIMUM number of points at which RofL can be evaluated
c is MAXPT.
  INTEGER MAXPT
  PARAMETER (MAXPT=6000)
  DOUBLE PRECISION Rp(MAXPT),Yp(MAXPT)
```

```
      COMMON/BLOCK100/Rp,Yp



c NUMPT indicates the number of points (Yp,Rp) is taken
      INTEGER NUMPT
      COMMON/BLOCK101/NUMPT



c Each level has a certain number of ZEROS stored in the
c array ZERO.ZN will be the maximum number of levels
c allowed and ZMAX is the maximum number of zeros for
c a given level.
      INTEGER ZN,ZMAX,INTER
      PARAMETER(ZN=NN,ZMAX=30,INTER=2)


      INTEGER MMAX,MINTER
      PARAMETER(MMAX=2*ZMAX,MINTER=2)


c
      integer   azero, nWcol, nYcol
      parameter (azero=0, nWcol=5, nYcol=2)
c
      integer   lower, upper, mean1, ms, height
      parameter (lower=1, upper=2, mean1=3, ms=4, height=5)
c
      double precision Winfo(azero:NNS,nWcol), Yinfo(1:NNS,nYcol)
      double precision ll1, ul1, ll2, ul2, yy
      common /select/ Winfo, Yinfo, ll1, ul1, ll2, ul2, yy
c
      integer igtype, s1, s2
      common /iselct/ igtype, s1, s2
c
      integer   corr, jpr, unipr
      parameter (corr=1, jpr=2, unipr=3)
c
      integer   cndprb, cndexp, convol
      parameter (cndprb=1, cndexp=2, convol=3)
c
      double precision relerr
      common/block26/relerr

c firspt is the first point at which RofL is evaluated
c lastpt is the last point at which RofL is evaluated
      double precision firstpt,lastpt
      common/block27/firstpt,lastpt

c constant MC
      double precision MC,varx,varz
      common/block28/MC,varz
```

```
c                       MAINU.F

c .....MAIN........MAIN.......MAIN.........MAIN.......MAIN

c This is the main program.First the program
c calls the routine INITDI, this routine provides the
c information of the different distributions that are
c needed and stored in arrays available to other routines.
c    1)If 0 is entered the algorithm is run with a number of
c      NIT iterations,if we want to continue where we left off
c      (after NIT iterations),the last partition generated by
c      the program is made available in file "start1".
c    2)If 1 or 2 is entered the program will find the LLoyd-Max
c      partition for sensor 1 or 2 those partitions are available
c      in "start1" or "start2".
c This is the following steps taken by MAIN :
c         1) The initial partition is stored in the array AR
c            using ReadAR(AR).
c         2) The routine SN(AR,C,MSE) is used to compute the
c            optimal coefficients for a given partition
c            and are stored in the array C.This routine
c            also computes the mean square error stored in MSE.
c         3) To find a better partition we iterate a
c            a fixed number of times (NIT).After each
c            iteration an updated partition along with its
c            optimal coefficients is computed using UL(COND,AR)
c            and SN(AR,C,MSE) respectively.
c
c The program will used two input files.
c         1) distr.dat ---> Contains the distribution informations
c         2) init.dat  ---> Indicates NS,N,SUBP,RELERR,NIT,TEST and
c                           DELTA respectively.
  PROGRAM main

  INCLUDE 'define.f'
  INCLUDE 'Carray.f'
  INCLUDE 'ARarray.f'
  INCLUDE 'MEANarray.f'
  INCLUDE 'NUarray.f'
  INCLUDE 'TEMP2array.f'

  INTEGER COUNT,TEST,I,J,K,IA,NIT,NUMBPT,NPT,KL
  DOUBLE PRECISION Yx,DUM,X1,X2,Y1,Y2,SUBP,MSE
  DOUBLE PRECISION MSI,DELTA,RofL,PT
  EXTERNAL ClearAR,ReadAR,Ul,MSI,LOAD,SN,INITDI
  EXTERNAL CreateRofL,RofL



  OPEN (UNIT=81,FILE='init.dat',STATUS='old')
  READ(81,*) NS,N,SUBP,RELERR
  READ(81,*) NIT,TEST,DELTA
  READ(81,*) MC
  CLOSE (UNIT=81)
  WRITE(*,*) 'Please Enter the Number of ITERATIONS .'
  WRITE(*,*) 'NIT=', NIT
c READ(*,*) NIT
  WRITE(*,*)
  MAX=2*SUBP
  WRITE(*,*) 'The Maximum number of Sensors is ',NNS
  WRITE(*,*) 'The Maximum number of Partitions is ',NN
  WRITE(*,*) 'The Maximum number of Subpartitions is ',NSUBP
  WRITE(*,*) 'The Number of Sensors is ',NS
  WRITE(*,*) 'The Number of Partitions is ',N
  WRITE(*,*)
```

```
      CALL INITDI

      WRITE(*,*)
      WRITE(*,*)'ENTER 0 to RUN the ALGORITHM'
      WRITE(*,*)'ENTER 1 to obtain the LLOYD-MAX for SENSOR 1'
      WRITE(*,*)'ENTER 2 to obtain the LLOYD-MAX for SENSOR 2'
      WRITE(*,*) 'TEST=', TEST
    c READ(*,*) TEST
      X1=Winfo(ZERO,MS)
      X2=Winfo(TEST,MS)
      Y1=Winfo(ZERO,MEAN1)
      Y2=Winfo(TEST,MEAN1)
      IF (TEST.EQ.0) THEN
              LMAX=.FALSE.
              VARX=X1

      ELSE
              LMAX=.TRUE.
              COND=TEST
              NS=1
              VARX=X1 + X2 + 2*Y1*Y2
      END IF
      A1=NS*N
      B1=A1*A1
      CALL ClearAR(AR)
      CALL ReadAR (AR)


      WRITE(*,*) ' The RELERR is:',RELERR
      CALL SN(AR,C,MSE)
      WRITE(*,*) 'The initial MEAN SQUARE ERROR is: ',MSE
      WRITE(*,*)
      DO 11  K=1,NS
      If (LMAX) THEN
         KL=TEST
      ELSE
         KL=K
      END IF
      X1=Yinfo(KL,LOWER)
      X2=Yinfo(KL,UPPER)
      WRITE(*,14) KL
      WRITE(*,12) KL,X1,X2
      11      CONTINUE
      WRITE(*,13) DELTA
      14      FORMAT('For Sensor',I2,':')
      12      FORMAT(3X,'Rof',I1,'(Y) sampled for: ',F7.4,' < Y <',F7.4)
      13      FORMAT('The INCREMENT DELTA used for both cases is:',F8.5,/)
      COUNT=0
              DO 55 Ia=1,NIT
      COUNT=COUNT+1
      WRITE(*,*) '*********',count,' ITERATION**************'

      DO 15 K=1,NS
         IF (LMAX) THEN
            COND=TEST
            SNUMB=1
         ELSE
            COND=k
                  SNUMB=k
         END IF
         J=1
         X1=Yinfo(COND,LOWER)
         X2=Yinfo(COND,UPPER)
         PT=(X2-X1)/DELTA
```

```
      NUMBPT=DINT(PT)
      NPT=NUMBPT +1
      OPEN(UNIT=21,FILE='out1',STATUS='UNKNOWN')
c     WRITE(*,*) 'Data for RofL is being computed..'
      DO 10 I=1,NPT
          Yx=X1 + DELTA*(I-1)
          IF ((Yx.GT.X1).AND.(Yx.LT.X2)) THEN
              Rp(J)=RofL(Yx)
              Yp(J)=YX
              IF (J.LE.MAXPT) THEN
                WRITE(21,32) Yx,Rp(J),J,I
   32                   FORMAT(F13.6,2x,F13.6,2x,I4,2x,I4)
              ELSE
                WRITE(*,*) 'ERROR IN DIMENSION FOR Rp ARRAY'
              END IF
              J=J+1
          END IF
   10       CONTINUE
c     write(*,*) 'ENTER ANY NUMBER TO  ,NTINUE.'
c     read(*,*) DUM

      CLOSE (UNIT=21)
      NUMPT=J-1
      CALL Ul(SNUMB,AR)
   15      CONTINUE

   CALL SN(AR,C,MSE)
   WRITE(*,*) 'The MEAN SQUARE ERROR is: ',MSE
   WRITE(*,*)

   55      CONTINUE
   STOP
   END
```

```
c                GEN.F


c         ....read data of partition into a 3-D arrray AR..
c This routine read from a file "start" the data of a given
c partition and store into a 3-D array AR.The index I of the
c matrix AR represent the sensor number,the index J represent the
c the subpartition number for a given sensor I finally the indices
c K and K+1 indicate the lower and upper values respectively of an
c interval of the subpartition J of a sensor I.A subpartition J of
c a given sensor I can be a union of intervals.When reading from
c "start" each subpartition J is separated by the two consecutive
c  numbers of -1.

  SUBROUTINE ReadAR (AR )

  INCLUDE 'define.f'

  DOUBLE PRECISION AR(NNS,NN,NMAX),LL,UL
  INTEGER I,J,K

  OPEN (UNIT=10,FILE='start',STATUS='old')
  DO 10 I=1,NS
          DO 20 J=1,N
             K=1
  30                 CONTINUE
             READ(10,*) LL,UL
                  AR(I,J,K)=LL
                  AR(I,J,K+1)=UL
                  K=K+2
                   IF ((UL.EQ.-1.0D0).AND.(LL.EQ.-1.0D0)) THEN
                  AR(I,J,K-2)=0.D0
                  AR(I,J,K-1)=0.D0
                  GO TO 20
             END IF
          GO TO 30
  20              CONTINUE
  10      CONTINUE
  RETURN
  END


c
c       .........INITIALIZE AR TO ZERO...........
  SUBROUTINE ClearAR (AR)

  INCLUDE 'define.f'

  INTEGER I,J,K
  DOUBLE PRECISION AR(NNS,NN,NMAX)

  DO 100 I=1,NS
           DO 200 J=1,N
              DO 300 K=1,MAX
           AR(I,J,K)=0.0D0
  300           CONTINUE
  200        CONTINUE
  100      CONTINUE
  RETURN
  END


C       This routine returns the matrix NU(I,J).For example if
c N(I,J)=10 it means that a subpartition J of a given sensor
c I is formed from the union of five different intervals.
```

```
c This matrix will be used
  SUBROUTINE COUNTP (NU)

  INCLUDE 'define.f'
  INCLUDE 'ARarray.f'

  INTEGER NU(MNS,MM)
          INTEGER L,J,KB


  DO 20 L=1,NS
     DO 30 J=1,N
        NU(L,J)=0
        KB=1
  10            CONTINUE
        IF (AR(L,J,KB).NE.AR(L,J,KB+1)) THEN
          IF (KB.LT.MAX) THEN
           NU(L,J)=NU(L,J) +2
                 KB=KB+2
           GO TO 10
                END IF
              END IF
  30          CONTINUE
  20       CONTINUE
  RETURN
  END


c Given two subpartitions Aki and Alj E[I (Yk )I (Yl)] is computed
c as follows:
c          1- If k=l and i=j then
c             P(Yk  Aki) is computed.Since Aki the subpartition
c             I of a sensor K can be a union of intervals the
c             probability of Yk along each interval is computed
c             and then summed together to perform this correctly
c             the number of intervals was needed (NU).
c             P(Yk Aki) was computed using PG.
c          2- If k=l and i NE j then
c             the routine is assigned the value ZERO
c          3- If k NE l then
c             P(Yk  Aki,Yl  Alj) is computed.
c             P(Yk  Aki,Yl  Alj) was computed using JPG.
c E[I (Yk )I (Yl)] is computed and stored in the matrix
c SUMP(L,J,K,I), this matrix is needed to compute the coefficients
c contained in the matrix C.

c   COMPUTE SUM OF PROBABILITIES IF WE HAVE UNION OF INTERVALS...

  DOUBLE PRECISION FUNCTION SUMP(L,J,K,I)

  INCLUDE 'define.f'
  INCLUDE 'ARarray.f'
  INCLUDE 'NUarray.f'

  DOUBLE PRECISION RES,ANS
  INTEGER NU1,NU2,NX,MX,I,J,K,L
  DOUBLE PRECISION JPG,PG
  EXTERNAL JPG,PG

  NU1=NU(L,J)
  NU2=NU(K,I)
  RES=0.0D0
  IF (L.EQ.K) THEN
     IF (I.EQ.J) THEN
          DO 15 NX=1,NU1,2
            ANS=PG(AR(L,J,NX),AR(L,J,NX+1),L)
```

```
            RES=RES+ANS
15            CONTINUE
          SUMP=RES
      ELSE
          SUMP=0.D0
      END IF
   ELSE
      DO 30 MX=1,NU1,2
         DO 40 NX=1,NU2,2
            ANS=JPG(AR(L,J,MX),AR(L,J,MX+1),AR(K,I,NX),
     *           AR(K,I,NX+1),L,K)
            RES=RES+ANS
40          CONTINUE
30       CONTINUE
         SUMP=RES
   END IF
   RETURN
   END


c This routine will compute E[XI (Yk)] the integral of
c fyk(y)*E[X/Yk=Y] over the subpartition Aki.It will use
c MEANX a routine that computes the integral of fyk(y)*E[X/Yk]
c over a given interval. Note that Aki is formed of a union of
c intervals ,knowing those intervals (AR),their number (NU) and
c using MEANX E[XI (Yk)] is computed and stored in the matrix
c SUMM(I,J) index I indicating the sensor and J the subpartition
c SUMM is needed to compute the coefficients contained in the
c matrix C.
c
c ..........sum of means...........

   DOUBLE PRECISION FUNCTION SUMM(I,J)

   INCLUDE 'define.f'
   INCLUDE 'ARarray.f'
   INCLUDE 'NUarray.f'

   INTEGER NU1,NX,I,J
   DOUBLE PRECISION RES,ANS
   DOUBLE PRECISION MEANX
   EXTERNAL MEANX

   NU1=NU(I,J)
   RES=0.0D0
   DO 15 NX=1,NU1,2
      ANS=MEANX(AR(I,J,NX),AR(I,J,NX+1),I)
          RES=RES+ANS
15    CONTINUE
   SUMM=RES
   RETURN
   END


c .............generate the R matrix.............
c This routine will generate the R matrix (a symmetric matrix)
c needed to solve the optimal coefficients.
c The equation that is needed to be solved is:
c                  Rc=mean
c where c is the matrix that contains the optimal coefficients.
   SUBROUTINE GENR (R)

   INCLUDE 'define.f'
   INCLUDE 'ARarray.f'
   INCLUDE 'NUarray.f'
   INCLUDE 'TEMP2array.f'
```

```
      INTEGER I1,J1,I,J,K,L
      DOUBLE PRECISION R(AS,AS)
      DOUBLE PRECISION SUMP
      EXTERNAL SUMP

      I1=0
      DO 10 L=1,NS
            DO 20 J=1,N
         I1=I1+1
         J1=0
            DO 30 K=1,NS
               DO 40 I=1,N
                     J1=J1+1
                     IF (J1.GE.I1) THEN
                         R(I1,J1)=SUMP(L,J,K,I)
                         IF (R(I1,J1).NE.0.0d0) THEN
c                            write(*,100) R(I1,J1),I1,J1
c  100                                FORMAT (F16.12, '  R(',I3,',',I3,')')
                         END IF
                     ELSE
                         R(I1,J1)=R(J1,I1)
                     END IF
      40            CONTINUE
      30         CONTINUE
      20      CONTINUE
      10   CONTINUE
      RETURN
      END


c     ........generate mean matrix.................
c This routine will generate the other matrix needed
c to solve for the optimal coefficients the mear natrix.
c The equation that is needed to be solved is:
c                  Rc=mean
c where c is the matrix that contains the optimal
c coefficients.

      SUBROUTINE GENM (MEAN)

      INCLUDE 'define.f'
      INCLUDE 'ARarray.f'
      INCLUDE 'NUarray.f'

      INTEGER IND,J,L
      DOUBLE PRECISION MEAN(AS)
      DOUBLE PRECISION SUMM
      EXTERNAL SUMM

      IND=1
      DO 10 L=1,NS
            DO 20 J=1,N
           MEAN(IND)=SUMM (L,J)
c          write(*,100) MEAN(IND),IND,l,j
c 100              format(F16.6,'  MEAN(',I3,')',2I3)
           IND=IND+1
      20    CONTINUE
      10   CONTINUE
      WRITE(*,*)
      RETURN
      END


c     .............SOLVE LINEAR EQT.................
c This routine will solve the equation: Rc=Mean
```

```
c c will be the matrix that contains the optimal
c coefficient.To obtain those coefficients the
c SVD approach was used.
c The NAG routine F02WEF used performs on R
c the matrix factorization known as the singular value
c decomposition,SOLVE also computes the minimized mean
c square error stored in MSE.


      SUBROUTINE SOLVE (R,MEAN,C,MSE)

      INCLUDE 'define.f'

      INTEGER LDR,LDM,LDPT,NCOLM
      PARAMETER (LDR=AS,LDM=AS,LDPT=AS,NCOLM=1)
      INTEGER LWORK
      PARAMETER(LWORK=AS**2 + 5*(AS-1))
      INTEGER I,IFAIL,J,IND
      LOGICAL WANTP,WANTQ


      DOUBLE PRECISION R(LDR,AS),MEAN(AS),C(NNS,NN),C1(AS),DUMMY(1)
      DOUBLE PRECISION WORK(LWORK),SV(AS),PT(LDPT,AS),MEANT(LDM)
      DOUBLE PRECISION RES,MSE,TAU,S,F06EAF
      EXTERNAL F06EAF,F02WEF

      DO 30 I=1,A1
         MEANT(I)=MEAN(I)
      30      CONTINUE
      UNIQUE=.TRUE.
      WANTQ=.TRUE.
      WANTP=.TRUE.
      IFAIL=0
      CALL F02WEF (A1,A1,R,LDR,NCOLM,MEANT,LDM,WANTQ,DUMMY,1,SV,
         *    WANTP,PT,LDPT,WORK,IFAIL)
      IF (IFAIL.NE.0) THEN
      write(*,*) ' ERROR IN SVD DECOMPOSITION'
      END IF
      OPEN (UNIT=1,FILE='svd.dat',STATUS='unknown')
      WRITE(1,11)
      WRITE(1,101) (SV(I),I=1,A1)
      WRITE(1,12)
      DO 10 I=1,A1
c     WRITE(1,101) (PT(J,I),J=1,A1)
      10      CONTINUE
      WRITE(1,13)
      DO 20 I=1,A1
c     WRITE(1,101) (R(I,J),J=1,A1)
      20      CONTINUE
      WRITE(1,14)
      WRITE(1,101) (MEANT(I),I=1,A1)
      CLOSE (UNIT=1)
c     TAU is the absolute error tolerance
      TAU=RELERR*SV(A1)
      DO 65 I=1,A1
         IF (SV(I).LT.TAU) GOTO 75
            NSV=I
      65      CONTINUE
      75      CONTINUE
      write(*,*) NSV,' singular values are >=',TAU
      write(*,*) 'SVMIN= ',SV(1)
      write(*,*) 'SVLAST= ',SV(NSV)
      write(*,*) 'SVMAX= ',SV(A1)
      DO 50 I=1,A1
           C1(I)=0.0D0
           DO 40 J=1,NSV
```

```
          S=MEANT(J)/SV(J)
          C1(I)=C1(I) +PT(J,I)*S
 40            CONTINUE
 50       CONTINUE
c WRITE(*,*) 'The solution to the system is:'
c WRITE(*,100) (C1(I),I=1,A1)


c Dot Product of the two vectors C1 and MEAN
  RES=F06EAF(AS,C1,1,MEAN,1)
  MSE=VARX-RES

  IND=1
  DO 15 I=1,NS
           DO 25 J=1,N
          C(I,J)=C1(IND)
          IND=IND+1
 25        CONTINUE
 15     CONTINUE
 100    FORMAT(4(1X,D12.4))
 11       FORMAT(/' SINGULAR VALUE')
 12       FORMAT(/' RIGHT-HAND SINGULAR VECTORS BY COLUMN')
 13       FORMAT(/' LEFT-HAND SINGULAR VECTORS BY COLUMN')
 14       FORMAT(/' VECTOR Q''*MEAN ')
 101    FORMAT(5(1X,D12.4))
  RETURN
  END


c         This routine generates the matrix c that contains
c the optimal coefficients given a partition A.
c This routine is used in the MAIN program.

  SUBROUTINE SN(AR,C,MSE)


  INCLUDE 'define.f'
  INCLUDE 'MEANarray.f'
  INCLUDE 'NUarray.f'

  DOUBLE PRECISION R(AS,AS),C(NNS,NN),AR(NNS,NN,NMAX),MSE
  EXTERNAL COUNTP,GENM,GENR,SOLVE

  CALL COUNTP(NU)
  CALL GENM(MEAN)
  CALL GENR(R)
  CALL SOLVE(R,MEAN,C,MSE)
  RETURN
  END


c .........sum of conditional prob...........
c This routine is needed to compute Rl(Y).
c It computes the conditional probabilities for
c a given subpartition.

  DOUBLE PRECISION FUNCTION SUMCP(I,J,Y)

  INCLUDE 'define.f'
  INCLUDE 'ARarray.f'
  INCLUDE 'NUarray.f'

  INTEGER I,J,IND
  DOUBLE PRECISION RES,ANS,Y
  INTEGER NU1,NX
  DOUBLE PRECISION FUN4
```

```
      EXTERNAL FUN4

      NU1=NU(I,J)
      RES=0.0D0
      IND=COND
      DO 15 NX=1,NU1,2
         ANS=FUN4(AR(I,J,NX),AR(I,J,NX+1),Y,I,IND)
c           write(*,*) 'ar(',i,',',j,',',nx,')', ar(i,j,nx)
c           write(*,*) 'ar(',i,',',j,',',nx+1,')', ar(i,j,nx+1)
               RES=RES+ANS
      15    CONTINUE
      SUMCP=RES
c           Write(*,*) res,'  res '
      RETURN
      END


c This routine is needed to compute Rl(Y).

      DOUBLE PRECISION FUNCTION CP (Y)

      INCLUDE 'define.f'
      INCLUDE 'Carray.f'
      INCLUDE 'ARarray.f'
      INCLUDE 'NUarray.f'


      INTEGER IND,J,K
      DOUBLE PRECISION Y
      DOUBLE PRECISION RES,ANS
      DOUBLE PRECISION SUMCP
      EXTERNAL SUMCP

        IND=SNUMB
        RES=0.0D0
        DO 10 K=1,NS
           IF (K.NE.IND) THEN
                    DO 20 J=1,N
              ANS=SUMCP (K,J,Y)*C(K,J)
              RES=ANS+RES
c             write(*,*) c(k,j), sumcp(k,j,y),' c(k,j)'
      20            CONTINUE
           END IF
      10     CONTINUE
        CP=RES
c     write(*,*) res,' fres'
      RETURN
      END


c ..............  Rl(Y).........................
c This routine computes Rl(y).

      DOUBLE PRECISION FUNCTION RofL(Y)

      INCLUDE 'define.f'
      INCLUDE 'ARarray.f'
      INCLUDE 'Carray.f'
      INCLUDE 'NUarray.f'


      INTEGER IND
      DOUBLE PRECISION Y
      DOUBLE PRECISION CP,EC
      EXTERNAL CP,EC
```

```
      IND=COND
c     IF (LMAX) THEN
          RofL=Y
c     ELSE
          RofL=EC(IND,Y) -CP(Y)
c     END IF
      RETURN
      END


c.....................RofLlev.............................

      DOUBLE PRECISION FUNCTION RofLlev(Y)

      INCLUDE 'define.f'

      DOUBLE PRECISION Y,RofL
      EXTERNAL RofL

      RofLlev=RofL(Y) - LEVEL

      RETURN
      END
```

```
c                ROOTFB.F

c This function finds one of the simple roots of R(Y) given
c the interval [ll,ul]. If more than one root exists in the
c         given interval an error message is issued.
c The subroutine C05ADF from the NAG library is used.

  DOUBLE PRECISION FUNCTION ROOT(LL,UL,RofL)

  DOUBLE PRECISION LL,UL,RofL,EPS,ETA,X
  EXTERNAL RofL,C05ADF
  INTEGER IFAIL

  EPS=1.0D-8
  ETA=0.0D0
  IFAIL=1
  CALL C05ADF(LL,UL,EPS,ETA,RofL,X,IFAIL)
  IF (IFAIL.GT.0) THEN
      WRITE(*,*) IFAIL ,' ERROR CHECK C05DAF in ROOT'
  END IF
  ROOT=X
  RETURN
  END


c This routine computes the different levels.
c The assumption in this algorithm is that C(l,1)<....<C(l,N)
c this is why the different elements of the n X N matrix C
c are sorted in a increasing order before the computation of
c the corresponding levels.

  SUBROUTINE LEVELS (LEV)

  INCLUDE 'define.f'
  INCLUDE 'Carray.f'

  INTEGER DUM,K,J,IND,I
  DOUBLE PRECISION LEV(NN),C(NNS,NN),TEMP

  DUM=N-1
  IND=SNUMB
  DO 15 K=1,N-1
      DO 25 J=N,K+1,-1
          IF (C(IND,J).LT.C(IND,J-1)) THEN
                  TEMP=C(IND,J-1)
                  C(IND,J-1)=C(IND,J)
                  C(IND,J)=TEMP
          END IF
  25          CONTINUE
  15      CONTINUE
  DO 10 I=1,DUM
          LEV(I)=(C(IND,I+1)+C(IND,I))/2.0D0
  10    CONTINUE
c write(*,*)
  write(*,*) 'The levels used are: '
  write(*,100) (lev(i),i=1,dum)
  100     format(4F16.6)
  write(*,*)
  RETURN
  END


c This routine computes all the zeros of Rp(Y) at a given
c         level lev(I).The zeros are stored in the matrix ZERO also the
c nearest value of Rp(Y) to the left and to the right of t
c the zero is stored in the matrix TESTPT,finally the
```

```
c number of zeros are stored in NUMOFZERO.
c To find the zeros the routine scans a list of data that
c contains values of Rp(Y)-lev(I) in search of a change of
c sign or a zero value.

  SUBROUTINE LISTOFZERO (ZERO,NUMOFZERO,TESTPT,LEV)

  INCLUDE 'define.f'

  INTEGER I,J,K,MAXPT
  INTEGER NUMOFZERO(ZN)
  DOUBLE PRECISION TESTPT(ZN,ZMAX,INTER),LEV(NN),ZERO (ZN,ZMAX)
  DOUBLE PRECISION ROOT,CHECK
  DOUBLE PRECISION RofLlev
  EXTERNAL LEVELS,ROOT
  EXTERNAL RofLlev



  DO 10 I=1,N-1
     K=1
     LEVEL=LEV(I)
     DO 20 J=1,NUMPT-1
          CHECK=(Rp(J)-LEV(I))*(Rp(J+1)-LEV(I))
          IF (CHECK.LT.0.0D0) THEN
              ZERO(I,K)=ROOT(Yp(J),Yp(J+1),RofLlev)
              TESTPT(I,K,1)=Rp(J)
              TESTPT(I,K,2)=Rp(J+1)
              K=K+1
          ELSE IF (Rp(J).EQ.LEV(I)) THEN
                  IF (J.NE.1) THEN
                      GO TO 20
                  ELSE
                      ZERO(I,K)=Yp(1)
                      TESTPT(I,K,1)=LEV(I)-Rp(J+1)
                      TESTPT(I,K,2)=Rp(J+1)
                      K=K+1
                  END IF
           ELSE IF  (Rp(J+1).EQ.LEV(I)) THEN
                  ZERO(I,K)=Yp(J+1)
                  TESTPT(I,K,1)=Rp(J)
                  TESTPT(I,K,2)=Rp(J+2)
                  K=K+1
          END IF
   20        CONTINUE
     NUMOFZERO(I)=K-1
   10     CONTINUE
  RETURN
  END


c This routine will merge in order two lists of zeros and
c testpoints.The list of zeros obtained at the level LEVA is
c merged with the list of zeros obtained in LEVB in an increasing
c order,the new list of zeros is stored in the array MZERO.
c Similarly the testpoints obtained at the two different levels
c are stored in increasing order in MTESTPT.

  SUBROUTINE MERGEANDSORT (LEVA,LEVB,ZERO,TESTPT,MZERO,MTESTPT,
     *  NUMOFZERO)

  INCLUDE 'define.f'

  DOUBLE PRECISION ZERO(ZN,ZMAX),TESTPT(ZN,ZMAX,INTER)
  DOUBLE PRECISION MZERO(MMAX),MTESTPT(MMAX,MINTER)
  INTEGER NUMOFZERO(ZN)
```

```
      INTEGER LEVA,LEVB,ILA,ILB,IM,T


      ILA=1
      ILB=1
      IM=1

      10      CONTINUE
      IF (ILA.LE.NUMOFZERO(LEVA)) THEN
          IF (ILB.LE.NUMOFZERO(LEVB)) THEN
              IF (ZERO(LEVA,ILA).LE.ZERO(LEVB,ILB)) THEN

                  MZERO(IM)=ZERO(LEVA,ILA)
                  MTESTPT(IM,1)=TESTPT(LEVA,ILA,1)
                  MTESTPT(IM,2)=TESTPT(LEVA,ILA,2)
                  ILA=ILA+1

              ELSE

                  MZERO(IM)=ZERO(LEVB,ILB)
                  MTESTPT(IM,1)=TESTPT(LEVB,ILB,1)
                  MTESTPT(IM,2)=TESTPT(LEVB,ILB,2)
                  ILB=ILB+1

              END IF
              IM=IM+1
              GO TO 10
          END IF
      END IF
      IF (ILA.GT.NUMOFZERO(LEVA)) THEN
          DO 20 T=ILB,NUMOFZERO(LEVB)
                MZERO(IM)=ZERO(LEVB,T)
                MTESTPT(IM,1)=TESTPT(LEVB,T,1)
                MTESTPT(IM,2)=TESTPT(LEVB,T,2)
                IM=IM+1
      20          CONTINUE
      ELSE
          DO 30 T=ILA,NUMOFZERO(LEVA)
                MZERO(IM)=ZERO(LEVA,T)
                MTESTPT(IM,1)=TESTPT(LEVA,T,1)
                MTESTPT(IM,2)=TESTPT(LEVA,T,2)
                IM=IM+1
      30          CONTINUE
      END IF
      RETURN
      END


c This routine given a list of the zeros and its testpoints
c obtained at a given level gives the new partition stored in
c the matrix ARS.The testpoints will be used to take a "peak"
c at the left or right of a zero which will enable us to
c determine if the range of r(y) in an interval formed by two
c consecutive zeros is above or below a certain level
c or between two levels.

      SUBROUTINE NEWPART (ZERO,TESTPT,INDX,NUMOFZERO,ARS,LEV)

      INCLUDE 'define.f'

      INTEGER INDX,NUM,L,LEVA,LEVB,I,J,K
      DOUBLE PRECISION ZERO(ZN,ZMAX),TESTPT(ZN,ZMAX,INTER)
      INTEGER NUMOFZERO(ZN)
      DOUBLE PRECISION ARS(NNS,NN,NMAX),LEV(NN)
      DOUBLE PRECISION MZERO(MMAX),MTESTPT(MMAX,MINTER)
      EXTERNAL LEVELS,MERGEANDSORT
```

```
      L=INDX
      DO 10 I=1,N
         K=1

c     We are searching for all the values of Y that belongs to
c     [-B-A,B+A] such that Rl(Y) is less than LEV(1).

         IF (I.EQ.1) THEN
            DO 20 J=1,NUMOFZERO(1)
               IF (TESTPT(I,J,1).LT.LEV(1)) THEN
                   IF (J.EQ.1) THEN
                       ARS(L,I,K)= ZERO(I,J)+2
                       ARS(L,I,K+1)=ZERO(I,J)
                       write(*,100) ars(l,i,k),ars(l,i,k+1),k
                       K=K+2
                   ELSE
                       ARS(L,I,K)=ZERO(I,J-1)
                       ARS(L,I,K+1)=ZERO(I,J)
                       write(*,100) ars(l,i,k),ars(l,i,k+1),k
                       K=K+2
                   END IF
               END IF
      20           CONTINUE
            IF (TESTPT(I,NUMOFZERO(1),2).LT.LEV(1)) THEN
                     ARS(L,I,K)=ZERO(I,J)
                     ARS(L,I,K+1)=ZERO(I,J)-3
                     write(*,100) ars(l,i,k),ars(l,i,k+1),k
                     K=K+2
            END IF

c     We are searching for all the values of Y that belongs to
c     [-B-A,B-A] such that Rl(Y) is greater than LEV(N-1).

         ELSE IF (I.EQ.N) THEN
            DO 30 J=1,NUMOFZERO(N-1)
               IF (TESTPT(I-1,J,1).GT.LEV(N-1)) THEN
                   IF (J.EQ.1) THEN
                       ARS(L,I,K)=ZERO(I-1,J)+2
                       ARS(L,I,K+1)= ZERO(I-1,J)
                       write(*,100) ars(l,i,k),ars(l,i,k+1),k
                       K=K+2
                   ELSE
                       ARS(L,I,K)=ZERO(I-1,J-1)
                       ARS(L,I,K+1)=ZERO(I-1,J)
                       write(*,100) ars(l,i,k),ars(l,i,k+1),k
                       K=K+2
                   END IF
               END IF
      30           CONTINUE
            IF (TESTPT(I-1,NUMOFZERO(N-1),2).GT.LEV(N-1)) THEN
                     ARS(L,I,K)=ZERO(I-1,NUMOFZERO(N-1))
                     ARS(L,I,K+1)=ZERO(I-1,NUMOFZERO(N-1))-3
                     write(*,100) ars(l,i,k),ars(l,i,k+1),k
                     K=K+2
            END IF

c     We are searching for all the values of Y that belongs to
c     [-B-A,B-A] such that Rl(Y) is between two given levels
c     LEVA and LEVB. MERGESORT is called to obtain the new lists
c     of zeros and testpoints.

         ELSE
            LEVA=I-1
            LEVB=I
```

```
      CALL MERGEANDSORT (LEVA,LEVB,ZERO,TESTPT,
     *          MZERO,MTESTPT,NUMOFZERO)
          NUM=NUMOFZERO(I-1)+NUMOFZERO(I)
      DO 40 J=1,NUM
         IF ((MTESTPT(J,1).GT.LEV(I-1)).AND.
     *                (MTESTPT(J,1).LT.LEV(I))) THEN
            IF (J.EQ.1) THEN
                ARS(L,I,K)=MZERO(J)+2
                ARS(L,I,K+1)=MZERO(J)
                write(*,100) ars(l,i,k),ars(l,i,k+1),k
                K=K+2
            ELSE
                ARS(L,I,K)=MZERO(J-1)
                ARS(L,I,K+1)=MZERO(J)
                write(*,100) ars(l,i,k),ars(l,i,k+1),k
                K=K+2
            END IF
         END IF
40          CONTINUE
      IF ((MTESTPT(NUM,2).GT.LEV(I-1)).AND.
     *            (MTESTPT(NUM,2).LT.LEV(I))) THEN
                ARS(L,I,K)=MZERO(NUM)
                ARS(L,I,K+1)=MZERO(NUM)-3
                write(*,100) ars(l,i,k),ars(l,i,k+1),k
                K=K+2
      END IF
  END IF
WRITE(*,*)
WRITE(*,*)
10      CONTINUE
100   FORMAT(2E15.5,I4)
RETURN
END


c.................copy ars into ar.........................
c The new contents stored in ARS is loaded into AR.
c If TEST=0 then after each iteration the new partition
c is copied into the file "start2".
c If TEST=1 or 2 then after each iteration the new
c partition is copied into "start1" or "start2"

  SUBROUTINE UPDATE (ARS,AR)

  INCLUDE 'define.f'

  INTEGER DEL,I,J,K,REF
  DOUBLE PRECISION ARS(NNS,NN,NMAX),AR(NNS,NN,NMAX)


  IF (NS.LE.2) GO TO 10
  WRITE(*,*) ' ERROR NEED TO CHANGE "UPDATE" IN ROOTFB.F.'
  10      CONTINUE
  I=SNUMB
     DO 20 J=1,N
        DO 30 K=1,MAX
                AR(I,J,K)=ARS(I,J,K)
  30          CONTINUE
  20       CONTINUE
  IF (COND.EQ.1) THEN
          OPEN(UNIT=12,FILE='start1',STATUS='UNKNOWN')
          REF=12
          IF (LMAX) THEN
          WRITE(*,*) 'THE NEW PARTITION IS LOADED INTO "start1".'
          END IF
  ELSE
```

```
        OPEN(UNIT=99,FILE='start2',STATUS='UNKNOWN')
        REF=99
        WRITE(*,*) 'THE NEW PARTITION IS LOADED INTO "start2".'
 END IF

 DEL=-1
   DO 15 I=1,NS
      DO 25 J=1,N
         DO 35 K=1,MAX,2
              IF (AR(I,J,K).NE.AR(I,J,K+1)) THEN
                 WRITE(REF,120) AR(I,J,K),AR(I,J,K+1)
                 WRITE(REF,130) DEL,DEL
 120                     FORMAT(F15.9,3X,F15.9)
 130                     FORMAT(2I6)
              END IF
 110                FORMAT(F16.11)
 35              CONTINUE
 25           CONTINUE
 15        CONTINUE
 CLOSE (UNIT=REF)
 RETURN
 END


c This routine will clear the following matrices:
c ZERO,TESTPT,MZERO,MTESTPT,NUMOFZERO.

  SUBROUTINE CLEAR (ZERO,TESTPT,MZERO,MTESTPT,NUMOFZERO)

  INCLUDE 'define.f'

  INTEGER I,J,K
  DOUBLE PRECISION ZERO(ZN,ZMAX),TESTPT(ZN,ZMAX,INTER)
  INTEGER NUMOFZERO(ZN)
  DOUBLE PRECISION MZERO(MMAX),MTESTPT(MMAX,MINTER)

  DO 10 I=1,ZN
     DO 20 J=1,ZMAX
          ZERO(I,J)=0.0D0
 20          CONTINUE
 10       CONTINUE

  DO 100 I=1,ZN
     DO 200 J=1,ZMAX
          DO 300 K=1,INTER
                  TESTPT(I,J,K)=0.0D0
 300              CONTINUE
 200         CONTINUE
 100      CONTINUE

  DO 15 I=1,ZN
          NUMOFZERO(I)=0.0D0
 15       CONTINUE

  DO 11 I=1,MMAX
     DO 21 J=1,MINTER
          MTESTPT(I,J)=0.0D0
 21          CONTINUE
 11       CONTINUE
  DO 14 I=1,MMAX
          MZERO(I)=0.0D0
 14       CONTINUE
  RETURN
  END
```

```
c.....................Ul(INDX,AR).............................
c Find new partition and store it in the matrix AR.
c This routine will be called in the MAIN PROGRAM.


  SUBROUTINE Ul(INDX,AR)

  INCLUDE 'define.f'

  INTEGER INDX
  DOUBLE PRECISION AR(NNS,NN,NMAX),ARS(NNS,NN,NMAX)
  DOUBLE PRECISION ZERO(ZN,ZMAX),TESTPT(ZN,ZMAX,INTER)
  INTEGER NUMOFZERO(ZN)
  DOUBLE PRECISION MZERO(MMAX),MTESTPT(MMAX,MINTER)
  EXTERNAL ClearAR,CLEAR,LISTOFZERO,NEWPART,UPDATE

  CALL ClearAR (ARS)
  CALL CLEAR (ZERO,TESTPT,MZERO,MTESTPT,NUMOFZERO)
  CALL LEVELS(LEV)
  CALL LISTOFZERO (ZERO,NUMOFZERO,TESTPT,LEV)
  CALL NEWPART (ZERO,TESTPT,INDX,NUMOFZERO,ARS,LEV)
  CALL UPDATE (ARS,AR)
  RETURN
  END
```

```
c                JOHN.F

c INITIALIZATION OF ARRAYS USING SUBROUTINE: INITDI
c
c     ****  THIS ROUTINE IS CHANGED DEPENDING ON THE EXAMPLE CHOSEN  ****
c
  subroutine initdi
c
  integer dist
  double precision a,b
c
c  Initialize common blocks for distributions.
c
  include 'define.f'
c
c  Read data from file
c
  open(unit=3,status='old',name='distr.dat')
  do 10 dist = azero,ns
     read (3,*) Winfo(dist,lower), Winfo(dist,upper)
     a=Winfo(dist,lower)
     b=Winfo(dist,upper)
     Winfo(dist,mean1)=(b+a)/2.0d0
     Winfo(dist,ms)=((b-a)**2)/12.0d0 + Winfo(dist,mean1)**2
   10     continue
  close (unit=3)


c
c  Print info. and compute heights for uniform distributions.
c
  print *, 'Underlying-distribution information:'
c
  do 20 dist = azero,ns
     Winfo(dist,height)
     &          = 1.d0/(Winfo(dist,upper)-Winfo(dist,lower))
     IF (DIST.EQ.0) THEN
        WRITE(*,11) Winfo(dist,lower),
     &          Winfo(dist,upper), Winfo(dist,height)
  11             FORMAT(' X :',F8.5,',',F8.5,',',F8.5)
     ELSE
        WRITE(*,12) dist, Winfo(dist,lower),
     &          Winfo(dist,upper), Winfo(dist,height)
  12             FORMAT('Z',I1,' :',F8.5,',',F8.5,',',F8.5)
     END IF
   20     continue
c
c print *, 'Computed observation-distribution information:'
c
  do 30 dist = 1,ns
     Yinfo(dist,lower) = Winfo(dist,lower) + Winfo(azero,lower)
     Yinfo(dist,upper) = Winfo(dist,upper) + Winfo(azero,upper)
c    print *, dist, ' :  ', Yinfo(dist,lower), Yinfo(dist,upper)
   30     continue
c
  return
  end


c
c  CINTGD function
c
c  This function returns the required integrand for computing
c
c     prob( Y_s1 in (a1,b1] | Y_s2 = yy ) * f_(Y_s2)(y)
c                                        (set igtype=cndprb=1)
```

```
c
c     E[ X | Y_s2 = yy ] * f_(Y_s2)(yy)      (set igtype=cndexp=2)
c
c     density for Y_s2                       (set igtype=convol=3)
c
      double precision function cintgd(x)
      include 'define.f'
      double precision densty, PW, x, z, l, u
c
c  Integrand for computing density of Y_s2, which is the convolution
c  of the density of X = W_azero with the density of W_s2.
c
      l = yy-x
      z = densty(azero,x) * densty(s2,l)
c
      goto ( 110, 120, 130 ), igtype
c
c  Integrand for computing prob( Y_s1 in (a1,b1] | Y_s2 = yy )
c
  110     l = ll1-x
      u = ul1-x
      z = z * PW(s1,l,u)
      goto 130
c
c  Integrand for computing E[ X | Y_s2 = yy ]
c
  120     z = z * x
c
  130     cintgd = z
c
      return
      end



c
c  FUNCTION UINTGD, set up integrand for nonconditional "stuff".
c
c     ****  THIS ROUTINE IS CHANGED DEPENDING ON THE EXAMPLE CHOSEN  ****
c
c
c  This function returns the required integrand for computing
c
c     E[ X I_(ll1,ul1](Y_s1) ]                    (set igtype=corr=1)
c
c     prob( Y_s1 in (ll1,ul1] , Y_s2 in (ll2,ul2] )  (set igtype=jpr=2)
c
c     prob( Y_s1 in (ll1,ul1] )                   (set igtype=unipr=3)
c
      double precision function uintgd(x)
      include 'define.f'
      double precision densty, PW, x, z,l,u,ll,ul,a,b,zplus
      logical novoid
      external inters
c
c  Integrand for computing prob( Y_s1 in (ll1,ul1] ).
c
      l = ll1-x
      u = ul1-x
      z = densty(azero,x) * PW(s1,l,u)
c
      goto ( 110, 120, 130 ), igtype
c
c  Integrand for computing E[ X I_(ll1,ul1](Y_s1) ]
c
  110     if (lmax) then
```

```
      a=winfo(s1,lower)
      b=winfo(s1,upper)
      call inters(a,b,l,u,ll,ul,novoid)
      if (novoid) then
          zplus= winfo(s1,height)*.5*(ul**2-ll**2)*densty(azero,x)
      else
          zplus=0.0d0
      end if
  else
            zplus = 0.0d0
  end if
        z = z * x + zplus
  goto 130
c
c  Integrand for computing prob( Y_s1 in (ll1,ul1] , Y_s2 in (ll2,ul2] )
c
  120     l = ll2-x
  u = ul2-x
  z = z * PW(s2,l,u)
c
  130     uintgd = z
  return
  end



c
c     ****  THIS ROUTINE IS CHANGED DEPENDING ON THE EXAMPLE CHOSEN  ****
c
c  Compute densty(dist,x), where dist=0 implies X
c  and dist>=1 implies W_dist
c
  double precision function densty(dist,x)
  double precision x
  integer dist
  include 'define.f'
c
  densty = Winfo(dist,height)
  return
  end


c     ****  THIS ROUTINE IS CHANGED DEPENDING ON THE EXAMPLE CHOSEN  ****
c
c  Compute prob( W_dist    (a,b] )
c
  double precision function PW(dist,a,b)
        double precision a, b, l, u, llim, ulim, z
  integer dist
        include 'define.f'
c
  if ( b .lt. a ) then
     z = 0
  else
     llim = Winfo(dist,lower)
     ulim = Winfo(dist,upper)
     if ( b .lt. llim ) then
          z = 0
     else if ( a .gt. ulim ) then
          z = 0
     else
          if ( a .lt. llim ) then
             l = llim
          else
             l = a
          endif
```

```
            if ( b .gt. ulim ) then
               u = ulim
            else
               u = b
            endif
            z = Winfo(dist,height)*(u - l)
         endif
      endif
      PW = z
      return
      end



c
c  Determine the intersection of [a,b] and [c,d].  Set novoid = .true.
c  if the intersection is nonempty.  Set novoid = .false. otherwise.
c  If novoid = .true. then [l,u] is the intersection of [a,b] and [c,d].
c
c
      SUBROUTINE inters(a,b,c,d,l,u,novoid)

      double precision  a,b,c,d,l,u
      logical novoid
c
      if (b .lt. a) then
            novoid= .false.
      else if (d .lt. c) then
            novoid= .false.
      else if ( d .lt. a ) then
            novoid = .false.
      else if ( c .gt. b ) then
            novoid = .false.
      else
            novoid = .true.
            if ( d .lt. b ) then
                  u = d
            else
                  u = b
            endif
            if ( c .gt. a ) then
                  l = c
            else
                  l = a
            endif
      endif
      return
      end



c This routine decodes an interval
      SUBROUTINE FIXUP (LLX,ULX,LL,UL,MAXL,MAXU)

      DOUBLE PRECISION CHECK
      DOUBLE PRECISION LLX,ULX,LL,UL,MAXU,MAXL

      CHECK=ULX-LLX

c [a,+INF]  ----->   [a,a-3] then check=-3.0d0  NB: +INF=MAXU
      IF (CHECK.LT.-2.6D0) THEN
          UL=MAXU
          LL=LLX
c [-INF,b]  ----->   [2+b,b] then check=-2.0d0 NB: -INF=MAXL
      ELSE IF (CHECK.LT.-1.6D0) THEN
```

```
      UL=ULX
            LL=MAXL
c [-INF,+INF]  ---->   [0,-1] then check=-1.0d0
  ELSE IF (CHECK.LT.-.5D0) THEN
      UL=MAXU
      LL=MAXL
c If check >0 then we have a finite interval.
  ELSE
      UL=ULX
      LL=LLX
  END IF
  RETURN
  END
```

```
c                JOHN2.F

c INITIALIZATION OF ARRAYS USING SUBROUTINE: INITDI
c
  subroutine initdi
c
  integer dist
  double precision a,b,First_Moment,Sec_Moment,lmse,alpha
  external First_Moment,Sec_Moment
c
c  Initialize common blocks for distributions.
c
  include 'define.f'
c
c  Read data from file
c
  open(unit=3,status='old',name='distr.dat')
  do 10 dist = azero,ns
     read (3,*) Winfo(dist,lower), Winfo(dist,upper)
     a=Winfo(dist,lower)
     b=Winfo(dist,upper)
     Winfo(dist,mean1)=First_Moment(b,dist)-First_Moment(a,dist)
     Winfo(dist,ms)=Sec_Moment(b,dist)-Sec_Moment(a,dist)-Winfo(dist,mean1)**2
   10     continue
  close (unit=3)


  a=Winfo(azero,ms)
  b=Winfo(1,ms)
  write(*,*) a,' and ',b,' are a and b'
  alpha=a/(a+b)
  lmse=a*(1.0d0-alpha)
  write(*,*)
  write(*,*) 'LMSE is equal: ',lmse
  write(*,*)

c
c  Print info. and compute heights for uniform distributions.
c
  print *, 'Underlying-distribution information:'
c
  do 20 dist = azero,ns
        WRITE(*,11) dist, Winfo(dist,lower),
     &          Winfo(dist,upper)
  11            FORMAT(I4,':',F8.5,',',F8.5)
   20     continue
c
c print *, 'Computed observation-distribution information:'
c
  do 30 dist = 1,ns
     Yinfo(dist,lower) = Winfo(dist,lower) + Winfo(azero,lower)
     Yinfo(dist,upper) = Winfo(dist,upper) + Winfo(azero,upper)
     WRITE(*,12) dist, Yinfo(dist,lower),
     &       Yinfo(dist,upper)
  12         FORMAT('Y',I4,':',F8.5,',',F8.5)
   30     continue
c
  return
  end



c
c  CINTGD function
c
c  This function returns the required integrand for computing
```

```
c
c     prob( Y_s1 in (a1,b1] | Y_s2 = yy ) * f_(Y_s2)(y)
c                                        (set igtype=cndprb=1)
c
c     E[ X | Y_s2 = yy ] * f_(Y_s2)(yy)      (set igtype=cndexp=2)
c
c     density for Y_s2                     (set igtype=convol=3)
c
      double precision function cintgd(x)
      include 'define.f'
      double precision densty, PW, x, z, l, u
c
c  Integrand for computing density of Y_s2; which is the convolution
c  of the density of X = W_azero with the density of W_s2.
c
      l = yy-x
      z = densty(azero,x) * densty(s2,l)
c
      goto ( 110, 120, 130 ), igtype
c
c  Integrand for computing prob( Y_s1 in (a1,b1] | Y_s2 = yy )
c
  110     l = ll1-x
      u = ul1-x
      z = z * PW(s1,l,u)
      goto 130
c
c  Integrand for computing E[ X | Y_s2 = yy ]
c
  120     z = z * x
c
  130     cintgd = z
c
      return
      end



c
c  FUNCTION UINTGD, set up integrand for nonconditional "stuff".
c
c  This function returns the required integrand for computing
c
c     E[ X I_(ll1,ul1](Y_s1) ]                    (set igtype=corr=1)
c
c     prob( Y_s1 in (ll1,ul1] , Y_s2 in (ll2,ul2] )  (set igtype=jpr=2)
c
c     prob( Y_s1 in (ll1,ul1] )                   (set igtype=unipr=3)
c
      double precision function uintgd(x)
      include 'define.f'
      double precision densty, PW, x, z,l,u,ll,ul,a,b,zplus,First_Moment
      logical novoid
      external inters,First_Moment
c
c  Integrand for computing prob( Y_s1 in (ll1,ul1] ).
c
      l = ll1-x
      u = ul1-x
      z = densty(azero,x) * PW(s1,l,u)
c
      goto ( 110, 120, 130 ), igtype
c
c  Integrand for computing E[ X I_(ll1,ul1](Y_s1) ]
c
```

```
110     if (lmax) then
        a=winfo(s1,lower)
        b=winfo(s1,upper)
        call inters(a,b,l,u,ll,ul,novoid)
        if (novoid) then
            Zplus=(First_Moment(b,s1)-First_Moment(a,s1))*densty(azero,x)
        else
            zplus=0.0d0
        end if
    else
                zplus = 0.0d0
    end if
            z = z * x + zplus
    goto 130
c
c  Integrand for computing prob( Y_s1 in (ll1,ul1] , Y_s2 in (ll2,ul2] )
c
120     l = ll2-x
    u = ul2-x
    z = z * PW(s2,l,u)
c
130     uintgd = z
c
    return
    end



c
c  Compute densty(dist,x), where dist=0 implies X
c  and dist>=1 implies W_dist
c
    double precision function densty(dist,x)

    include 'define.f'
    double precision x,X01aaf,PI,alpha,b,v,heights
    integer dist
    external x01aaf
c
    PI=X01aaf(x)
    a=Winfo(dist,lower)
    b=Winfo(dist,upper)
    v=(3.0d0*PI)/(2.0d0*b)
    alpha=(6.0d0*PI)/((15.0d0*PI+8.0d0)*b)
          heights= 1.d0/(b-a)

    If (dist.eq.1) then
      densty= alpha*(5.0d0/4.0d0 - cos(v*x))
c     densty = heights
    else if (dist.eq.azero) then
c     densty = heights
      densty= alpha*(5.0d0/4.0d0 - cos(v*x))
    else
c     densty = heights
      densty= alpha*(5.0d0/4.0d0 - cos(v*x))
    end if

    return
    end
c
c  Compute prob( W_dist in (a,b] )
c
    double precision function PW(dist,a,b)
          double precision a, b, l, u, llim, ulim, z,Distr_Func
    integer dist
          include 'define.f'
```

```
      external Distr_Func
c
      if ( b .lt. a ) then
         z = 0
      else
         llim = Winfo(dist,lower)
         ulim = Winfo(dist,upper)
         if ( b .lt. llim ) then
              z = 0
         else if ( a .gt. ulim ) then
              z = 0
         else
              if ( a .lt. llim ) then
                 l = llim
              else
                 l = a
              endif
              if ( b .gt. ulim ) then
                 u = ulim
              else
                 u = b
              endif

              z= Distr_Func(u,dist)-Distr_Func(l,dist)

         endif
      endif
      PW = z
      return
      end



c
c  Determine the intersection of [a,b] and [c,d].  Set novoid = .true.
c  if the intersection is nonempty.  Set novoid = .false. otherwise.
c  If novoid = .true. then [l,u] is the intersection of [a,b] and [c,d].
c
c
      SUBROUTINE inters(a,b,c,d,l,u,novoid)

      double precision  a,b,c,d,l,u
      logical novoid
c
      if (b .lt. a) then
              novoid= .false.
      else if (d .lt. c) then
              novoid= .false.
      else if ( d .lt. a ) then
              novoid = .false.
      else if ( c .gt. b ) then
              novoid = .false.
      else
              novoid = .true.
              if ( d .lt. b ) then
                      u = d
              else
                      u = b
              endif
              if ( c .gt. a ) then
                      l = c
              else
                      l = a
              endif
      endif
```

```
      return
      end



      SUBROUTINE FIXUP (LLX,ULX,LL,UL,MAXL,MAXU)

      DOUBLE PRECISION CHECK
      DOUBLE PRECISION LLX,ULX,LL,UL,MAXU,MAXL

      CHECK=ULX-LLX

c [a,+INF]  ---->   [a,a-3] then check=-3.0d0  NB: +INF=MAXU
      IF (CHECK.LT.-2.6D0) THEN
          UL=MAXU
          LL=LLX
c [-INF,b]  ---->   [2+b,b] then check=-2.0d0 NB: -INF=MAXL
      ELSE IF (CHECK.LT.-1.6D0) THEN
          UL=ULX
                LL=MAXL
c [-INF,+INF]  ---->   [0,-1] then check=-1.0d0
      ELSE IF (CHECK.LT.-.5D0) THEN
          UL=MAXU
          LL=MAXL
c If check >0 then we have a finite interval.
      ELSE
          UL=ULX
          LL=LLX
      END IF
      RETURN
      END



      double precision function Sec_Moment (X,point)

      include 'define.f'
      integer point
      double precision x,alpha,v,b,PI,x01aaf,a,heights
      external x01aaf

      pi=X01aaf(x)
      a=Winfo(point,lower)
      b=Winfo(point,upper)
      v=(3*PI)/(2*b)
      alpha=(6.0d0*PI)/((15.0d0*PI+8.0d0)*b)
            heights= 1.d0/(b-a)

      If (point.eq.azero) then
c       Sec_Moment=((X**3)*heights)/3.0d0
        Sec_Moment=((5.0d0/12.0d0)*X**3-(1/v)**3*(v*x*cos(v*x)-2.0d0*sin(v*x)
     &                  +v**2*x**2.0d0*sin(v*x)))*Alpha
      else if (point.eq.1) then
c       Sec_Moment=((X**3)*heights)/3.0d0
        Sec_Moment=((5.0d0/12.0d0)*X**3-(1/v)**3*(v*x*cos(v*x)-2.0d0*sin(v*x)
     &                  +v**2*x**2.0d0*sin(v*x)))*Alpha
      else
c       Sec_Moment=((X**3)*heights)/3.0d0
        Sec_Moment=((5.0d0/12.0d0)*X**3-(1/v)**3*(v*x*cos(v*x)-2.0d0*sin(v*x)
     &                  +v**2*x**2.0d0*sin(v*x)))*Alpha
      end if

      return
      end
```

```
      double precision function First_Moment (X,point)

      include 'define.f'
      integer point
      double precision x,alpha,v,b,PI,x01aaf,a,heights
      external x01aaf

      pi=X01aaf(x)
      a=Winfo(point,lower)
      b=Winfo(point,upper)
      v=(3.0d0*PI)/(2.0d0*b)
      alpha=(6.0d0*PI)/((15.0d0*PI+8.0d0)*b)
            heights= 1.d0/(b-a)

      If (point.eq.azero) then
c       First_Moment=(X**2)*.50d0*heights
        First_Moment=((5.0d0/8.0d0)*x**2 -(1/v)**2*(cos(v*x)+v*x*sin(v*x)))
     &                        *Alpha
      else if (point.eq.1) then
c       First_Moment=(X**2)*.50d0*heights
        First_Moment=((5.0d0/8.0d0)*x**2 -(1/v)**2*(cos(v*x)+v*x*sin(v*x)))
     &                        *Alpha
      else
c       First_Moment=(X**2)*.50d0*heights
        First_Moment=((5.0d0/8.0d0)*x**2 -(1/v)**2*(cos(v*x)+v*x*sin(v*x)))
     &                        *Alpha
      end if

      return
      end


      double precision function Distr_Func (X,point)

      include 'define.f'
      integer point
      double precision x,alpha,v,b,PI,x01aaf,a,heights
      external x01aaf

      pi=X01aaf(x)
      a=Winfo(point,lower)
      b=Winfo(point,upper)
      v=(3.0d0*PI)/(2.0d0*b)
      alpha=(6.0d0*PI)/((15.0d0*PI+8.0d0)*b)
            heights= 1.d0/(b-a)

      If (point.eq.azero) then
c       Distr_Func=(X)*heights
        Distr_Func=((5.0d0/4.0d0)*x -(1/v)*sin(v*x))*Alpha
      else if (point.eq.1) then
        Distr_Func=((5.0d0/4.0d0)*x -(1/v)*sin(v*x))*Alpha
c       Distr_Func=(X)*heights
      else
c       Distr_Func=(X)*heights
        Distr_Func=((5.0d0/4.0d0)*x -(1/v)*sin(v*x))*Alpha
      end if

      return
      end
```

```
c                COND.F

 DOUBLE PRECISION FUNCTION CONV (IND,Y)

 INCLUDE 'define.f'
 INCLUDE 'nag.f'
 INTEGER IND
 DOUBLE PRECISION Y,L,U,A,B,C,D,RES,CINTGD
 LOGICAL NOVOID
 EXTERNAL CINTGD,INTERS

 A=WINFO(AZERO,LOWER)
 B=WINFO(AZERO,UPPER)
 S2=IND
 YY=Y

 IF (YY.GT.YINFO(S2,LOWER) .AND. YY.LT.YINFO(S2,UPPER)) THEN
             C=YY-WINFO(S2,UPPER)
             D=YY-WINFO(S2,LOWER)
             CALL INTERS(A,B,C,D,L,U,NOVOID)
             IF (NOVOID) THEN
                 IGTYPE=CONVOL

                 EPSABS= 1d-10
                 EPSREL=1d-8
                 IFAIL=-1
                 CALL D01AJF(CINTGD,L,U,EPSABS,EPSREL,RES
    *                        ,ABSERR,W,LW,IW,LIW,IFAIL)
                 IF (IFAIL.NE.0) THEN
                         WRITE(*,*) 'IN CONV'
                         WRITE(*,*) 'LOWER LIMIT: ',L
                         WRITE(*,*) 'UPPER LIMIT: ',U
                 END IF
             ELSE
                 RES=0.0D0
              END IF
 ELSE
        RES=0.000
 END IF
 CONV=RES
 RETURN
 END


 DOUBLE PRECISION FUNCTION EC(CONDI,Y)

 INCLUDE 'define.f'
 INCLUDE 'nag.f'
 INTEGER CONDI
 DOUBLE PRECISION Y,L,U,A,B,C,D,RES,CONV,CINTGD
 LOGICAL NOVOID
 EXTERNAL CINTGD,CONV,INTERS

 A=WINFO(AZERO,LOWER)
 B=WINFO(AZERO,UPPER)
 S2=CONDI
 YY=Y

 IF (YY.GT.YINFO(S2,LOWER) .AND. YY.LT.YINFO(S2,UPPER)) THEN
        IF (LMAX) THEN
            RES=Y
        ELSE
            C=YY-WINFO(S2,UPPER)
            D=YY-WINFO(S2,LOWER)
            CALL INTERS(A,B,C,D,L,U,NOVOID)
            IF (NOVOID) THEN
```

```
                IGTYPE=CNDEXP

                EPSABS= 1d-10
                EPSREL=1d-8
                IFAIL=-1
                CALL D01AJF(CINTGD,L,U,EPSABS,EPSREL,RES
     *                     ,ABSERR,W,LW,IW,LIW,IFAIL)
                IF (IFAIL.NE.0) THEN
                       WRITE(*,*) 'IN EC.'
                       WRITE(*,*) 'LOWER LIMIT: ',L
                       WRITE(*,*) 'UPPER LIMIT: ',U
                END IF
                RES=RES/CONV(S2,YY)
            ELSE
                RES=0.0D0
             END IF
        END IF
ELSE
        RES=0.0D0
END IF
EC=RES
RETURN
END


DOUBLE PRECISION FUNCTION FUN4 (LLX,ULX,Y,IND,CONDI)

INCLUDE 'define.f'
INCLUDE 'nag.f'
INTEGER IND,CONDI
DOUBLE PRECISION Y,L,U,A,B,C,D,RES,CONV,CINTGD
DOUBLE PRECISION LLX,ULX,MAXL,MAXU
LOGICAL NOVOID
EXTERNAL CINTGD,CONV,INTERS,FIXUP


A=WINFO(AZERO,LOWER)
B=WINFO(AZERO,UPPER)
S1=IND
S2=CONDI
YY=Y
MAXU=YINFO(S2,UPPER)
MAXL=YINFO(S2,LOWER)
CALL FIXUP (LLX,ULX,LL1,UL1,MAXL,MAXU)

IF (YY.GT.YINFO(S2,LOWER) .AND. YY.LT.YINFO(S2,UPPER)) THEN
        C=YY-WINFO(S2,UPPER)
        D=YY-WINFO(S2,LOWER)
        CALL INTERS (A,B,C,D,L,U,NOVOID)
        IF (NOVOID) THEN
            IGTYPE=CNDPRB

            EPSABS= 1d-10
            EPSREL=1d-8
c           EPSREL=1d-6
            IFAIL=-1
            CALL D01AJF(CINTGD,L,U,EPSABS,EPSREL,RES
     *                 ,ABSERR,W,LW,IW,LIW,IFAIL)
            IF (IFAIL.NE.0) THEN
                WRITE(*,*) 'IN FUN4.'
                WRITE(*,*) 'LOWER LIMIT: ',L
                WRITE(*,*) 'UPPER LIMIT: ',U
                WRITE(*,*) 'THE IND and COND ARE: ',IND,COND
            END IF
            RES=RES/CONV(S2,YY)
        ELSE
```

```
            RES=0.000
        END IF
    ELSE
        RES=0.000
    END IF
    FUN4=RES
    RETURN
    END
```

```
c                UNCOND.F

 DOUBLE PRECISION FUNCTION PG (LLX,ULX,INDEX)

 INCLUDE 'nag.f'
 INCLUDE 'define.f'
 INTEGER INDEX
 DOUBLE PRECISION LLX,ULX,L,U,RES,MAXL,MAXU
 DOUBLE PRECISION UINTGD,A,B,C,D
 EXTERNAL UINTGD,FIXUP,INTERS
 LOGICAL NOVOID

 IF (LMAX) THEN
    S1=COND
 ELSE
   S1=INDEX
 END IF
 MAXU=YINFO(S1,UPPER)
 MAXL=YINFO(S1,LOWER)
 CALL FIXUP (LLX,ULX,LL1,UL1,MAXL,MAXU)
 A=WINFO(AZERO,LOWER)
 B=WINFO(AZERO,UPPER)
 C=LL1 - WINFO(S1,UPPER)
 D=UL1 - WINFO(S1,LOWER)
 CALL INTERS(A,B,C,D,L,U,NOVOID)
 IGTYPE=UNIPR
 IFAIL=-1
 EPSABS=1D-10
 EPSREL=1.0D-08


 IF (NOVOID) THEN
 CALL D01AJF(UINTGD,L,U,EPSABS,EPSREL,RES,ABSERR,
    *  W,LW,IW,LIW,IFAIL)
 IF (IFAIL.NE.0) THEN
         WRITE(*,*) 'IN PG'
         WRITE(*,*) 'LOWER LIMIT: ',L
         WRITE(*,*) 'UPPER LIMIT: ',U
 END IF
 ELSE
         RES=0.0D0
 END IF
 PG=RES
 RETURN
 END


 DOUBLE PRECISION FUNCTION JPG(LLX1,ULX1,LLX2,ULX2,IND1,IND2)

 INCLUDE 'nag.f'
 INCLUDE 'define.f'
 INTEGER IND1,IND2
 DOUBLE PRECISION LLX2,ULX2,LLX1,ULX1
 DOUBLE PRECISION UINTGD,L,U,RES,MAXL,MAXU
 DOUBLE PRECISION A,B,C,D,TEMPL,TEMPH
 EXTERNAL UINTGD,FIXUP,INTERS
 LOGICAL NOVOID

 S1=IND1
 S2=IND2
 A=WINFO(AZERO,LOWER)
 B=WINFO(AZERO,UPPER)
 MAXL=YINFO(S1,LOWER)
 MAXU=YINFO(S1,UPPER)
 CALL FIXUP (LLX1,ULX1,LL1,UL1,MAXL,MAXU)
 C= LL1 - WINFO(S1,UPPER)
```

```
      D= UL1 - WINFO(S1,LOWER)
      CALL INTERS(A,B,C,D,TEMPL,TEMPH,NOVOID)
      IF (NOVOID) THEN
        MAXL=YINFO(S2,LOWER)
        MAXU=YINFO(S2,UPPER)
        CALL FIXUP (LLX2,ULX2,LL2,UL2,MAXL,MAXU)
        C= LL2 - WINFO(S2,UPPER)
        D= UL2 - WINFO(S2,LOWER)
        CALL INTERS(TEMPL,TEMPH,C,D,L,U,NOVOID)
c       write(*,*) L,U
        IF (NOVOID) THEN
          IGTYPE=JPR
          IFAIL=-1
          EPSABS=1D-10
          EPSREL=1.0D-06
c         EPSREL=1.0D-08

          CALL D01AJF(UINTGD,L,U,EPSABS,EPSREL,RES,ABSERR,
     *                      W,LW,IW,LIW,IFAIL)
          IF (IFAIL.NE.0) THEN
              WRITE(*,*) 'IN JPG'
              WRITE(*,*) 'LOWER LIMIT: ',L
              WRITE(*,*) 'UPPER LIMIT: ',U
          END IF
          JPG=RES
        ELSE
          JPG=0.0D0
          GOTO 10
        END IF
      ELSE
        JPG=0.0D0
      END IF
   10       CONTINUE
      RETURN
      END



      DOUBLE PRECISION FUNCTION MEANX (LLX,ULX,INDEX)

      INCLUDE 'nag.f'
      INCLUDE 'define.f'
      INTEGER INDEX
      DOUBLE PRECISION LLX,ULX,L,U,MAXL,MAXU,RES
      DOUBLE PRECISION UINTGD,A,B,C,D
      LOGICAL NOVOID
      EXTERNAL UINTGD,FIXUP,INTERS


      IF (LMAX) THEN
         S1=COND
      ELSE
        S1=INDEX
      END IF
      MAXU=YINFO(S1,UPPER)
      MAXL=YINFO(S1,LOWER)
      CALL FIXUP (LLX,ULX,LL1,UL1,MAXL,MAXU)
      A=WINFO(AZERO,LOWER)
      B=WINFO(AZERO,UPPER)
      C=LL1 - WINFO(S1,UPPER)
      D=UL1 - WINFO(S1,LOWER)
c     write(*,*) C,d ' is c and d'
      CALL INTERS(A,B,C,D,L,U,NOVOID)
c     write(*,*) L,U
      IGTYPE=CORR
      IFAIL=-1
```

```
EPSABS=1D-10
EPSREL=1.0D-08


IF (NOVOID) THEN
CALL D01AJF(UINTGD,L,U,EPSABS,EPSREL,RES,ABSERR,
   *  W,LW,IW,LIW,IFAIL)
IF (IFAIL.NE.0) THEN
        WRITE(*,*) 'IN MEANX'
        WRITE(*,*) 'LOWER LIMIT: ',L
        WRITE(*,*) 'UPPER LIMIT: ',U
END IF
ELSE
        RES=0.000
END IF
MEANX=RES
RETURN
END
```

References

[1] S. P. Lloyd, "Least squares quantization in PCM", *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 129-137, M. r. 1982.

[2] J. Max, "Quantizing for minimum distortion", *IRE Trans. Inform. Theory*, vol. IT-6, pp. 7-12, Mar. 1960.

[3] G. E., Forsythe, *Computer Methods for Mathematical Computations.* New Jersey: Prentice-Hall, 1977

[4] J. A. Gubner, "Finite-dimensional nonlinear estimators for distributed estimation systems", *Book of Abstracts, 1990 IEEE International Symposium on Information Theory*, San Diego, CA, Jan. 1990.

[5] H. V. Poor, *An Introduction to Signal Detection and Estimation.* New York: Springer-Verlag, 1988.

[6] A. Papoulis, *Probability, Random Variables, and Stochastic Processes.* New York: McGraw-Hill, 1984.

Appendix C

Reprint of Reference [5]

On the Computation of Shot-Noise Probability Distributions

John A. Gubner, *Member, IEEE*

Abstract — A general method that does not use numerical integration is presented for approximating a continuous cumulative distribution function (cdf) from its characteristic function. If the cdf has a density that is piecewise smooth except for jump discontinuities, and if the discontinuity locations can be identified, then excellent density approximation can be obtained by fitting a cubic spline to the cdf approximation and then differentiating the spline. Since shot-noise densities typically contain essential discontinuities as well as impulses, the method is adapted to handle these complications.

Index Terms — Filtered point process, Poisson process, Fourier transform, Fourier series, fast Fourier transform, Gibbs phenomenon, sampling theorem, spline.

I. INTRODUCTION

Shot-noise processes, also known as filtered point processes, constitute an important class of mathematical models used to understand physical phenomena ranging from the measurement of nerve impulses in the brain, to the formation of images on film exposed under low-level illumination, to the electric current generated by photodiodes used in optical communication systems. Hence, it is unfortunate that in most cases, shot-noise densities must be obtained by numerical contour integration of their characteristic functions or other sophisticated techniques.

We first present a general method for recovering a continuous cumulative distribution function (cdf) from its characteristic function without numerical integration. If the cdf has a density $f(y)$ that is piecewise smooth except for jump discontinuities, and if the discontinuity locations can be identified, we can easily recover the density as well by fitting a cubic spline to the cdf approximation and then differentiating the spline. Unfortunately, shot-noise densities typically not only have an impulse at the origin but also satisfy $\lim_{y\to 0+} f(y) = \infty$. We must therefore study the characteristic function more carefully and modify the initial method to account for these complications.

The paper is organized as follows. In Section II we summarize the general method for the case of a piecewise-smooth density with jump discontinuities. In Section III we introduce our shot-noise model and summarize prior work on recovering shot-noise densities. In Section IV we analyze shot-noise characteristic functions and moment-generating functions. In Section V we modify the method of Section II to handle shot-noise random variables. Several examples are considered and density plots are presented. Since shot-noise characteristic functions themselves can also be difficult to compute, we consider an approximation scheme in Section VI.

Part of this research has been submitted to the 1993 IEEE International Symposium on Information Theory, San Antonio, TX, January 17-22, 1993. This work was supported in part by the Air Force Office of Scientific Research under Grants AFOSR-90-0181 and F49620-92-J-0305.

The author is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.

II. THE GENERAL METHOD

We propose the following general approach for recovering a continuous cdf from its characteristic function. The approach is based on the following lemma, which is proved in the Appendix. We then discuss a method for recovering the corresponding density, assuming that it is piecewise smooth.

Lemma 1: Let μ be a finite nonatomic measure on $\mathbb{R}$ with characteristic function

$$\Phi(\omega) := \int_{-\infty}^{\infty} e^{j\omega x} d\mu(x).$$

Then

$$\mu(0,\infty) = \lim_{L\to\infty} \sum_{n=-\infty}^{\infty} b_n \Phi(n\pi/L), \tag{1}$$

where

$$b_n = \begin{cases} 1/2, & n = 0, \\ -j/n\pi, & n = \text{odd}, \\ 0, & n = \text{even} \neq 0. \end{cases} \tag{2}$$

Furthermore, for $0 < L < \infty$, the error,

$$\sum_{n=-\infty}^{\infty} b_n \Phi(n\pi/L) - \mu(0,\infty),$$

is upper bounded by $\mu(-\infty, -L]$ and lower bounded by $-\mu(L,\infty)$.

If we now fix any $y \in \mathbb{R}$ and consider the measure $\mu_y(C) = \mu(C + y)$, it is an easy corollary that

$$\mu(y,\infty) = \lim_{L\to\infty} \sum_{n=-\infty}^{\infty} b_n \Phi(n\pi/L) e^{-jn\pi y/L}, \tag{3}$$

with the corresponding upper and lower error bounds being $\mu(-\infty, -L+y]$ and $-\mu(L+y,\infty)$. If we restrict attention to y in a subinterval $[L_1, L_2] \subset (-L, L)$, then

$$\mu(-\infty, -L+y] \leq \mu(-\infty, -L+L_2], \tag{4}$$

and

$$-\mu(L+y,\infty) \geq -\mu(L+L_1,\infty). \tag{5}$$

Note that if $\mu(-\infty,0)=0$, then the upper bound is zero, and if $\mu(0,\infty)=0$, then the lower bound is zero. To obtain more easily computable error bounds, we assume that

$$M(s) := \int_{-\infty}^{\infty} e^{sx}\,d\mu(x) < \infty, \quad \text{for all } s \in \mathbb{R}.$$

Then for $y \in [L_1, L_2]$ and $s > 0$, we have

$$\begin{aligned}\mu(L+y,\infty) &\leq \mu(L+L_1,\infty)\\ &\leq e^{-s(L+L_1)}M(s)\\ &\leq e^{-I_+(L+L_1)},\end{aligned}$$

where $I_+(x) := \sup_{s>0}[sx - K(s)]$ and $K(s) := \ln M(s)$. Similarly,

$$\begin{aligned}\mu(-\infty,-L+y] &\leq \mu(-\infty,-L+L_2]\\ &\leq e^{-I_-(-L+L_2)},\end{aligned} \tag{6}$$

where $I_-(x) := \sup_{s>0}[-sx - K(-s)]$. It is well known from the theory of large deviations that $I_\pm(x)$ is convex, continuous, and nonnegative. Furthermore, the suprema are achieved for some $s \geq 0$ [5, pp. 7-9].

Remark: Since Φ is the characteristic function of μ, equation (3) can be regarded as a sampling theorem for the approximately bandlimited signal $\Phi(\omega)$. If Φ were truly bandlimited to $(L_1, L_2]$; i.e., if $\mu(-\infty, L_1] = \mu(L_2, \infty) = 0$, then (3) would hold exactly for finite $L > L_2 - L_1$, and $2\pi/(L_2 - L_1)$ would be the Nyquist sampling period. When Φ is not truly bandlimited, taking $L < \infty$ will introduce aliasing, the effect of which can be bounded as discussed above.

To use the formula in (3), set

$$F^c_{L,N}(y) := \sum_{n=-N}^{N} b_n \Phi(n\pi/L) e^{-jn\pi y/L}. \tag{7}$$

Now let $c_0 := 0$ and $c_n := b_n\Phi(n\pi/L)$ for $n \neq 0$. Observe that c_{-n} is the complex conjugate of c_n and that $c_N = 0$ for N even. We can therefore write, for even N,

$$F^c_{L,N}(y) = b_0\Phi(0) + 2\,\mathrm{Re}\sum_{n=0}^{N-1} c_n e^{-jn\pi y/L}.$$

If we let $y = k\,\Delta y$, where $\Delta y = 2L/N$, then the samples $F^c_{L,N}(k\,\Delta y)$ for $k = 0, \ldots, N-1$ can be computed with an N-point fast Fourier transform (FFT). We then propose to fit a cubic spline [3] to the samples of $F^c_{L,N}(k\,\Delta y)$ that lie in the range $[L_1, L_2]$. This approximation is motivated by the following result.

Theorem (Birkhoff and de Boor [2]): Let F be a function that is four times continuously differentiable on an interval $[a,b]$. If F_m is a sequence of interpolating cubic splines satisfying the additional constraints $F'_m(a) = F'(a)$ and $F'_m(b) = F'(b)$, then F_m, F'_m, F''_m, and F'''_m converge uniformly to F, F', F'', and F''', respectively, assuming that as $m \to \infty$, (*i*) the number of knots of F_m tends to infinity, (*ii*) the maximum knot spacing tends to zero, and (*iii*) the ratios of the knot spacings of F_m are bounded.

If the measure μ has a piecewise-smooth density, we approximate it by differentiating (the negative of) the cubic spline approximation of $F^c_{L,N}$ between the knots. To obtain a good approximation, it is necessary to place triple knots at the density jump points, double knots at corners, and single knots at corners in the derivative of the density. This will become clearer in the examples.

Example 1: Let μ be the probability measure whose density $f(y)$ is one-half the sum of the uniform density on $[-.5, .5]$ and the standard normal density. The corresponding characteristic function is

$$\Phi(\omega) = \frac{1}{2}\left[\frac{\sin(\omega/2)}{\omega/2} + e^{-\omega^2/2}\right].$$

Since $\Phi(\omega)$ decays like $1/\omega$, and since b_n decays like $1/n$, we see that for fixed L, the series in (7) converges uniformly to a continuous function as $N \to \infty$, and thus the finite series will not exhibit Gibbs phenomenon. Of course, if we were to differentiate (7), the new series coefficients would decay like $1/n$, and Gibbs phenomenon would appear. We now take $L = 6$, $L_1 = 0$, and $L_2 = 3$. For the error bounds, it suffices to consider (6) since the density is symmetric. Now,

$$M(s) = \frac{1}{2}\left[\frac{e^{s/2} - e^{-s/2}}{s} + e^{s^2/2}\right].$$

It is easily verified numerically that the maximum value of $-s(-3) - \ln M(-s)$ is greater than 5.17 and occurs at $s \approx 3$. The bound is then $e^{-5.17} \leq .006$. We next took $N = 256$ and obtained 66 samples of $F^c_{L,N}$ in the range $[0, 3.02]$. We first used 30 uniformly spaced knots in the proper subinterval $[.0001, 3]$ to generate a cubic spline approximation of $F^c_{L,N}$ using the NAG library subroutines E02BAF and E02BCF. In Fig. 1 both $f(y)$ and minus the derivative of the spline are plotted. The oscillations at 1/2 are *not* due to the Gibbs phenomenon associated with Fourier series. Rather, they are due to the fact that a cubic spline with distinct knots is twice continuously differentiable, while $\lim_{N\to\infty} F^c_{L,N}(y)$ is not even continuous at $y = 1/2$. Based on Fig. 1, we fitted a new spline with only 8 knots, including a triple knot at 1/2 to handle the discontinuity in the density. The knot sequence was .0001, .5, .5, .5, 1, 1.5, 2, 3. The resulting approximation to $f(y)$

is shown if Fig. 2. It is interesting to compare Fig. 2 with the straightforward FFT-approximation obtained from

$$f(y) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\varphi(\omega)e^{-j\omega y}\,d\omega \approx \frac{1}{2\pi}\sum_{n=-(N-1)}^{N-1}\varphi(n\,\Delta\omega)e^{-jn\,\Delta\omega\,y}\Delta\omega$$

by taking $\Delta\omega = \pi/L$ and $y = k\,\Delta y$, where $\Delta y = (2\pi/N)/\Delta\omega = 2L/N$. With these substitutions,

$$f(k\,\Delta y) \approx \frac{\Delta\omega}{2\pi}\left(\varphi(0) + 2\,\mathrm{Re}\sum_{n=0}^{N-1} d_n e^{-j\frac{2\pi}{N}kn}\right), \qquad (8)$$

where $d_0 = 0$ and $d_n = \varphi(n\pi/L)$. Using $L = 6$ and $N = 256$, we obtained the graph shown in Fig. 3, which clearly shows the Gibbs phenomenon associated with the Fourier series of a function with a jump discontinuity.

III. Shot-Noise Models

Consider the real-valued process $\{Z_t\}$ given by

$$Z_t = \sum_{\nu} A_\nu h(t, T_\nu) + V_t, \qquad (9)$$

where the $\{T_\nu\}$ are points of a Poisson process with nonnegative intensity $\lambda(\cdot)$, $\{A_\nu\}$ is an independent, identically distributed, nonnegative "gain" sequence, and $\{V_t\}$ is a zero-mean Gaussian process. We assume $\{A_\nu\}$, $\{T_\nu\}$, and $\{V_t\}$ are mutually statistically independent. The deterministic function h is the system impulse response, or point spread function, depending on the application.

A. Applications of the Model

The model described by (9) is quite general. If t represents time, $\{Z_t\}$ can be used as a model of the photoelectric current generated in the receiver of an optical communication system. If t represents two-dimensional spatial position, then $\{Z_t\}$ can be used as a model of image formation on film.

1. Optical Communication Systems

In optical communication systems, Z_t models the electric current delivered by an amplifier whose input signal is the output of an avalanche photodiode. In this situation the $\{T_\nu\}$ denote the times at which arriving photons are detected in the receiver, the $\{A_\nu\}$ are the avalanche gains, h is the impulse response of the amplifier, and $\{V_t\}$ models thermal noise in the amplifier. The intensity $\lambda(\cdot)$ of the Poisson point process is determined by the brightness of the light falling on the photodiode.

For high light levels, Z_t is approximately normal (even if no Gaussian noise is added) [18, 19]. For low light levels, the Gaussian approximation would not be so accurate.

2. Image Detection

Consider x-ray images on film with point spread function $h(t, \tau), t, \tau \in \mathbb{R}^2$. Photons striking the film cause "smears" instead of well-defined points. The response of the film at a point $t \in \mathbb{R}^2$ due to a photon arriving at a point T_ν is $A_\nu h(t, T_\nu)$. This gives rise to the model

$$Z_t = \sum_{\nu} A_\nu h(t, T_\nu),$$

which is a special case of (9) in which the Gaussian noise is absent.

B. Summary of Prior Work

The most basic shot-noise statistics, namely the mean and variance, were reported by Campbell in 1909 [6, 7]. Shot noise was also investigated by Schottky in his 1918 paper on spontaneous current fluctuations in electric conductors [21]. In 1944–45, Rice [19] gave an extensive analysis of shot noise when the underlying Poisson process has a constant intensity. In particular, he showed that as the intensity tends to infinity, the probability distribution of the shot noise tends to a normal distribution. In 1971 Papoulis [18], considering underlying Poisson processes with time-varying intensity, gave numerical bounds on the difference between the true shot-noise distribution and the Gaussian approximation. The first nonasymptotic results concerning the cumulative distribution of shot noise appeared in the 1960 paper by Gilbert and Pollak [10]. For an underlying Poisson process with constant intensity, they derived an integral equation satisfied by the shot-noise distribution. They were able to solve this integral equation for some special cases of the shot-noise impulse response. Progress on the numerical computation of the density of shot noise was reported by Richter and Smits [20] in 1974. Their approach was to approximate the characteristic function by piecewise polynomial segments, which could then be inverse Fourier transformed in closed form. In 1975 Foschini *et al.* [9] gave a detailed analysis of the detection of a shot-noise signal in the presence of additive white Gaussian noise. They emplcyed several approximations in order to obtain manageable expressions for the likelihood function, from which they could gain insight into the general structure of the optimum detector. The 1976 paper of Mazo and Salz [16] analyzed the performance of integrate-and-dump filters. They also obtained exact formulas for the probability distribution of the $\{A_\nu\}$ in physical avalanche diodes. Further work on computing the density of shot noise appeared in the 1978 paper by Yue *et al.* [23]. Their approach was to approximate the tails of shot-noise densities by a weighted sum of normal densities. In 1984 Morris [17] considered an imaging model in which the photon locations $\{T_\nu\}$ were observed and passed through a linear

filter matched to the underlying Poisson-process intensity $\lambda(\cdot)$. He then studied hypothesis testing based on the resulting shot-noise random variable. In 1988 Kadota [14] reported approximately optimum detection of deterministic signals in Gaussian and compound Poisson noise. His model included a noise process that consisted of samples of a Gaussian process where the sample times were Poisson distributed. In 1990, Lowen and Teich [15] considered shot-noise processes with a power-law impulse response to obtain $1/f$ noise. They assumed that the underlying Poisson process had a constant intensity, and they showed that such shot-noise distributions need *not* converge to a normal law as the intensity increases. In 1991 Hero [13] approximated the likelihood ratio for observing a shot-noise process in the presence of additive white Gaussian noise. His approximation became more accurate as impulse response of the shot noise became more narrow and more closely resembled the underlying point process. Most recently, Helstrom and Ho [12] have successfully applied numerical contour integration to the inversion of the shot-noise characteristic function to obtain the shot-noise cumulative distribution in the case when additive Gaussian noise is present.

IV. Shot-Noise Characteristic Functions

In the remainder of the paper we fix t, set $g(\tau) := h(t, \tau)$, and focus on the random variable

$$Y := \sum_{\nu} A_\nu g(T_\nu).$$

The characteristic function of Y is [22, p. 170],

$$\varphi(\omega) := \mathsf{E}[e^{j\omega Y}] = \exp(\psi(\omega)),$$

where

$$\psi(\omega) := \int \lambda(\tau)[\varphi_A(\omega g(\tau)) - 1]\, d\tau, \qquad (10)$$

and φ_A is the common characteristic function of the $\{A_\nu\}$.

Remark: If we had added Gaussian noise in the definition of Y, then the characteristic function of Y would have decayed like $e^{-\sigma^2\omega^2/2}$, where σ^2 is the variance of the Gaussian noise. In this case, Y would have an infinitely differentiable density, and the method of Section II would apply.

The purpose of this section is to give conditions under which we can write $\psi(\omega) = -B + \tilde{\psi}(\omega)$, where $\tilde{\psi}(\omega) \to 0$ as $|\omega| \to \infty$. It will then follow that the density of Y contains an impulse of strength e^{-B} at the origin. We will also identify the constant B and the inverse Fourier transform of $\tilde{\psi}$.

As suggested by the $d\tau$ in (10), we are implicitly assuming that the underlying Poisson process lives on some finite-dimensional euclidean space. For temporal Poisson processes, the integral in (10) is over $\mathbb{R} = (-\infty, \infty)$. For two-dimensional spatial Poisson processes, we have a double integral over the euclidean plane $\mathbb{R}^2$. In any case, using Fubini's Theorem, we can rewrite (10) as

$$\psi(\omega) = \mathsf{E}\left[\int \lambda(\tau)[e^{j\omega g(\tau)A_\nu} - 1]\, d\tau\right], \qquad (11)$$

assuming that

$$\mathsf{E}\left[\int \lambda(\tau)|e^{j\omega g(\tau)A_\nu} - 1|\, d\tau\right] < \infty. \qquad (12)$$

In order that (12) be true even when A_ν is a constant random variable, we assume that

$$\int \lambda(\tau)|e^{j\omega g(\tau)} - 1|\, d\tau < \infty, \quad \text{for all } \omega \in \mathbb{R}. \qquad (13)$$

Remark: A sufficient condition for (13) to hold is that the integral of λ over the support of g be finite. For example, consider a temporal Poisson process with $\lambda(\tau) = 0$ for $\tau < 0$. Let h be a causal impulse response, i.e., $h(t, \tau) = 0$ for $\tau > t$. Then with $g(\tau) = h(t, \tau)$ for some fixed $t \geq 0$, the integral in (13) becomes

$$\int_0^t \lambda(\tau)|e^{j\omega h(t,\tau)} - 1|\, d\tau.$$

Clearly, this integral will be finite for every $t \geq 0$ and every $\omega \in \mathbb{R}$ assuming only that λ is locally integrable.

In view of (11), we set

$$\psi_0(\omega) := \int \lambda(\tau)[e^{j\omega g(\tau)} - 1]\, d\tau \qquad (14)$$

so that we can write

$$\psi(\omega) = \mathsf{E}[\psi_0(\omega A_\nu)]. \qquad (15)$$

To analyze (14), we proceed as follows. We first define two measures,

$$\Lambda(D) := \int_D \lambda(\tau)\, d\tau,$$

where D is any Borel subset of the euclidean space on which the underlying Poisson process lives, and

$$\Gamma(C) := \Lambda(\{\tau : g(\tau) \in C\}),$$

where C is any Borel subset of $\mathbb{R}$.

Remark: If $\Lambda(D) < \infty$ for some set D, then

$$\frac{\Lambda(\{\tau : g(\tau) \in C\} \cap D)}{\Lambda(D)} \qquad (16)$$

can be regarded as the conditional probability of the event $\{g(T_1) \in C\}$ given that exactly one Poisson-distributed point occurs in D. If $\{\tau: g(\tau) \in C\} \subset D$, then (16) reduces to $\Gamma(C)/\Lambda(D)$.

We now apply the change-of-variable formula for measures [4, p. 219, Theorem 16.12] to (14) to obtain

$$\begin{aligned}\psi_0(\omega) &= \int [e^{j\omega g(\tau)} - 1]\, d\Lambda(\tau) \\ &= \int_{-\infty}^{\infty} [e^{j\omega\theta} - 1]\, d\Gamma(\theta).\end{aligned}$$

Since the integrand is zero for $\theta = 0$, we can write

$$\psi_0(\omega) = \int_{-\infty}^{\infty} [e^{j\omega\theta} - 1]\, d\Gamma_0(\theta),$$

where Γ_0 is the measure defined by

$$\Gamma_0(C) := \Gamma(C\backslash\{0\}).$$

If Γ_0 is a finite measure, i.e., if

$$\Gamma_0(\mathbb{R}) = \int_{\{\tau:\, g(\tau)\neq 0\}} \lambda(\tau)\, d\tau < \infty,$$

then we can set $B_0 := \Gamma_0(\mathbb{R})$ and write

$$\psi_0(\omega) = -B_0 + \int_{-\infty}^{\infty} e^{j\omega\theta}\, d\Gamma_0(\theta).$$

If Γ_0 has a density γ_0, then we can write

$$\psi_0(\omega) = -B_0 + \int_{-\infty}^{\infty} e^{j\omega\theta}\gamma_0(\theta)\, d\theta. \tag{17}$$

Proposition 2: If Γ_0 is a finite measure with density γ_0, then

$$\psi(\omega) = -B + \int_{-\infty}^{\infty} e^{j\omega\theta}\gamma(\theta)\, d\theta,$$

where

$$B := B_0 \cdot \mathsf{P}(A_\nu > 0),$$

and

$$\gamma(\theta) := \mathsf{E}\left[\frac{\gamma_0(\theta/A_\nu)}{A_\nu}\mathbf{1}_{\{A_\nu>0\}}\right] \tag{18}$$

is integrable; the symbol $\mathbf{1}$ is the indicator function of the specified set.

Proof: First, since γ_0 is integrable, Tonelli's Theorem and a change of variable imply that γ is integrable. Next, substitute (17) into (15) and break up the expectation over the events $\{A_\nu = 0\}$ and $\{A_\nu > 0\}$. Make a change of variable and then use Fubini's Theorem. □

If we now set

$$\tilde{\psi}(\omega) := B + \psi(\omega) = \int_{-\infty}^{\infty} e^{j\omega\theta}\gamma(\theta)\, d\theta, \tag{19}$$

then $\tilde{\psi}(\omega) \to 0$ as $|\omega| \to \infty$ by the Riemann–Lebesgue Lemma. It then follows that $\varphi(\omega) \to e^{-B}$ as $|\omega| \to \infty$, and therefore the density of Y contains an impulse of strength e^{-B} at the origin.

An analogous development can be carried out for the moment-generating function of Y,

$$m(s) := \mathsf{E}[e^{sY}] = \exp(k(s)),$$

where for $s \in \mathbb{C}$, $k(s) := \mathsf{E}[k_0(sA_\nu)]$,

$$k_0(s) := \int \lambda(\tau)[e^{sg(\tau)} - 1]\, d\tau,$$

and we assume that for all $\sigma \in \mathbb{R}$,

$$\mathsf{E}\left[\int_{\{\tau:\, g(\tau)\neq 0\}} \lambda(\tau)e^{\sigma g(\tau)A_\nu}\, d\tau\right] < \infty.$$

Note that when $\sigma = 0$, this reduces to the weaker assumption that $\Gamma_0(\mathbb{R}) < \infty$.

A. Examples

Let $\|\cdot\|$ be a norm on $\mathbb{R}^2$. Fix any $r > 0$, and let q map $[0, r]$ onto $[0, 1]$, where q is nonnegative, continuous, and strictly decreasing so that $q^{-1}: [0, 1] \to [0, r]$ exists and is nonnegative, continuous, and strictly decreasing. Set $g(\tau) = q(\|\tau\|)$ for $\|\tau\| \le r$ and set $g(\tau) = 0$ for $\|\tau\| > r$. Since

$$g(\tau) > \theta \iff \|\tau\| < q^{-1}(\theta), \quad 0 \le \theta < 1,$$

the definition of Γ_0 yields

$$\Gamma_0(\theta, \infty) = \int_{D(\theta)} \lambda(\tau)\, d\tau, \quad 0 \le \theta < 1,$$

where

$$D(\theta) := \{\tau \in \mathbb{R}^2 : \|\tau\| < q^{-1}(\theta)\}.$$

Since g is nonnegative, $\Gamma_0(\theta, \infty) = B_0$ for $\theta < 0$, and since the maximum value of g is 1, $\Gamma_0(\theta, \infty) = 0$ for $\theta \ge 1$. The density of Γ_0 is $\gamma_0(\theta) = -(d/d\theta)\Gamma_0(\theta, \infty)$. We next let

$$\lambda(\tau) = u(\|\tau\|^2),$$

where $u: [0, \infty) \to [0, \infty)$. If $\|\cdot\|$ is the usual euclidean norm, then changing to polar coordinates yields

$$\Gamma_0(\theta, \infty) = 2\pi \int_0^{q^{-1}(\theta)} u(\rho^2)\rho\, d\rho, \quad 0 \le \theta \le 1.$$

If U is an antiderivative of u, then

$$\frac{d}{d\rho}U(\rho^2) = 2\rho u(\rho^2),$$

and

$$\Gamma_0(\theta,\infty) = \pi[U(q^{-1}(\theta)^2) - U(0)], \quad 0 \le \theta \le 1.$$

For example, the Gaussian intensity, $\lambda(\tau) = \exp(-\|\tau\|^2)$, corresponds to $u(\rho) = e^{-\rho}$ and $U(\rho) = -e^{-\rho}$. In this case,

$$\Gamma_0(\theta,\infty) = \pi(1 - e^{-q^{-1}(\theta)^2}), \quad 0 \le \theta \le 1.$$

The Cauchy intensity, $\lambda(\tau) = 1/(1+\|\tau\|^2)$, corresponds to $u(\rho) = 1/(1+\rho)$ and $U(\rho) = \ln(1+\rho)$, yielding

$$\Gamma_0(\theta,\infty) = \pi \ln(1 + q^{-1}(\theta)^2), \quad 0 \le \theta \le 1.$$

The constant intensity, $\lambda(\tau) \equiv 1$, corresponds to $u(\rho) = 1$ and $U(\rho) = \rho$, resulting in

$$\Gamma_0(\theta,\infty) = \pi q^{-1}(\theta)^2, \quad 0 \le \theta \le 1.$$

Example 2: Consider the piecewise quadratic function

$$q(x) = \begin{cases} 1 - ax^2, & 0 \le x \le z, \\ b(1-x)^2, & z \le x \le 1, \end{cases} \tag{20}$$

where $a > 0$, $b > 0$, and $0 \le z \le 1$ are given. If $0 < z < 1$, we see from (20) that $q(0) = 1$, $q(1) = 0$, and $q'(0) = q'(1) = 0$. If $a = b/(b-1)$ and $z = b/(a+b)$, then q will be continuously differentiable at z, and thus q will be a quadratic spline on $[0,1]$, an example of which is shown in Fig. 4. Now,

$$q^{-1}(\theta) = \begin{cases} 1 - \sqrt{\theta/b}, & 0 \le \theta \le q(z), \\ \sqrt{(1-\theta)/a}, & q(z) \le \theta \le 1, \end{cases}$$

and if $\lambda(\tau) \equiv 1$,

$$\Gamma_0(\theta,\infty) = \begin{cases} \pi(1 - 2\sqrt{\theta/b} + \theta/b), & 0 \le \theta \le q(z), \\ \pi(1-\theta)/a, & q(z) \le \theta \le 1, \end{cases}$$

and

$$\gamma_0(\theta) = \begin{cases} \pi((\theta/b)^{-1/2} - 1)/b, & 0 < \theta \le q(z), \\ \pi/a, & q(z) \le \theta \le 1. \end{cases}$$

Using (17), we find that for $\omega > 0$,

$$\psi_0(\omega) = \pi\left\{ \frac{e^{j\omega} - e^{j\omega q(z)}}{j\omega a} + \frac{1 - e^{j\omega q(z)}}{j\omega b} + 2\sqrt{\frac{q(z)}{b}} \cdot \frac{\mathrm{Fr}(x)}{x} - 1 \right\}, \tag{21}$$

where $x = \sqrt{2q(z)\omega/\pi}$, and

$$\mathrm{Fr}(x) := \int_0^x \exp(j\tfrac{\pi}{2}\theta^2)\, d\theta.$$

The real and imaginary parts of $\mathrm{Fr}(x)$ are Fresnel integrals, and are easily computed with the NAG library subroutines S20ADF and S20ACF, respectively. In addition, for $s > 0$, we can write

$$k_0(s) = \pi\left\{ \frac{e^s - e^{sq(z)}}{sa} + \frac{1 - e^{sq(z)}}{sb} + 2\frac{e^{sq(z)}}{\sqrt{sb}}\mathrm{Daw}(\sqrt{s\,q(z)}) - 1 \right\},$$

where

$$\mathrm{Daw}(x) := e^{-x^2}\int_0^x e^{\theta^2}\, d\theta$$

is Dawson's integral, and is easily computed with the NAG library subroutine S15AFF.

Example 3: Let q be as in (20) with $z = 1$ and $a = 1$. This results in

$$q^{-1}(\theta) = \sqrt{1-\theta}, \quad 0 \le \theta \le 1.$$

If $\lambda(\tau)$ is the Cauchy intensity mentioned earlier, then

$$\Gamma_0(\theta,\infty) = \pi \ln(2-\theta), \quad 0 \le \theta \le 1,$$

$$\gamma_0(\theta) = \pi/(2-\theta), \quad 0 \le \theta \le 1,$$

and for $\omega > 0$,

$$\psi_0(\omega) = -\pi \ln 2 + \pi \int_0^1 \frac{e^{j\omega\theta}}{2-\theta}\, d\theta = -\pi \ln 2 + \pi e^{j2\omega}\int_\omega^{2\omega} \frac{e^{-j\theta}}{\theta}\, d\theta.$$

The real and imaginary parts of this last integral are given by

$$\mathrm{Ci}(2\omega) - \mathrm{Ci}(\omega) \quad \text{and} \quad \mathrm{Si}(\omega) - \mathrm{Si}(2\omega),$$

respectively, where

$$\mathrm{Ci}(\omega) := \mathrm{C} + \ln\omega + \int_0^\omega \frac{\cos\theta - 1}{\theta}\, d\theta,$$

$\mathrm{C} \approx 0.5772$ is Euler's constant, and

$$\mathrm{Si}(\omega) := \int_0^\omega \frac{\sin\theta}{\theta}\, d\theta.$$

The functions Ci and Si are easily computed with the NAG library subroutines S13ACF and S13ADF, respectively. In addition, for $s > 0$,

$$k_0(s) = -\pi \ln 2 + \pi e^{2s}[E_1(s) - E_1(2s)],$$

where

$$E_1(x) := \int_x^\infty \frac{e^{-\theta}}{\theta}\, d\theta, \quad x > 0,$$

is the exponential integral, and is easily computed with the NAG library subroutine S13AAF.

Example 4 (Morris [17]): For $\tau = (x, y) \in \mathbb{R}^2$, set

$$g(\tau) = g(x,y) = \begin{cases} 1-|x|, & |x| \le 1, |y| \le 1, \\ 0, & \text{otherwise,} \end{cases}$$

and let $\lambda(\tau) = (\overline{N}_r/2)g(\tau)$. Then

$$\Gamma_0(\theta, \infty) = \overline{N}_r(1-\theta)(1+\theta), \quad 0 \le \theta \le 1,$$

and

$$\gamma_0(\theta) = 2\overline{N}_r\theta, \quad 0 \le \theta \le 1.$$

Note that $B_0 = \Gamma_0(\mathbb{R}) = \Gamma_0(0,\infty) = \overline{N}_r$. Using (17),

$$\psi_0(\omega) = B_0\left[-1 + \frac{e^{j\omega/2}}{\omega/2}\left(\sin(\omega/2) + j\frac{\sin(\omega/2)}{\omega/2} - j\cos(\omega/2)\right)\right], \qquad \omega \neq 0.$$

Also, for $s \neq 0$,

$$k_0(s) = B_0\left[-1 + 2\left(\frac{e^s}{s} - \frac{e^s - 1}{s^2}\right)\right].$$

Example 5: Let $q(x) = \exp(-px^2)$, where $p > 0$. If we regard q as mapping $[0,\infty)$ onto $(0,1]$, then

$$q^{-1}(\theta) = \sqrt{\frac{-\ln\theta}{p}}, \quad 0 < \theta \le 1.$$

If $\lambda(\tau)$ is the Gaussian intensity mentioned earlier, then

$$\Gamma_0(\theta,\infty) = \pi(1-\theta^{1/p}), \quad 0 \le \theta \le 1,$$

and

$$\gamma_0(\theta) = \frac{\pi}{p}\theta^{\frac{1}{p}-1}, \quad 0 < \theta \le 1.$$

Clearly, if $p = 1/n$ for some positive integer n, then on $[0,1]$, $\gamma_0(\theta)$ is a polynomial of degree $n-1$, and (17) can be evaluated in closed form using integration by parts. However, if $p > 1$, then $\gamma_0(0+)$ will not be finite; for example, if $p = 2$, γ_0 will exhibit $1/\sqrt{\theta}$ behavior as in Example 2.

Remark: If $z = 1$ and $a = 1$ in Example 2, and if $p = 1$ in Example 5, then the resulting γ_0 functions are the same, $\gamma_0(\theta) = \pi$. If $p = 1/2$ in Example 5, and if $\overline{N}_r = \pi$ in Example 4, then the γ_0 functions are again the same, $\gamma_0(\theta) = 2\pi\theta$.

V. Shot-Noise Cumulative Distributions and Densities

Let F denote the cumulative probability distribution of Y. We assume that F is continuous everywhere except the origin, where it has a jump discontinuity of size e^{-B}. Then the measure

$$\mu(C) := \mathsf{P}(Y \in C \text{ and } Y \neq 0) \tag{22}$$

is nonatomic, and

$$\begin{aligned} \Phi(\omega) &= \int_{-\infty}^{\infty} e^{j\omega x}\, d\mu(x) \\ &= \mathsf{E}[e^{j\omega Y}\mathbf{1}_{\{Y\neq 0\}}] \\ &= \varphi(\omega) - e^{-B} \\ &= e^{-B}[e^{\tilde{\psi}(\omega)} - 1]. \end{aligned}$$

Since $\tilde{\psi}(\omega) \to 0$, $\Phi(\omega) \approx e^{-B}\tilde{\psi}(\omega)$ for large $|\omega|$. Furthermore, $\tilde{\psi}$ can decay very slowly for shot-noise processes. Recalling Kummer's Comparison Method [11, p. 195], we write

$$\Phi(\omega) = e^{-B}[e^{\tilde{\psi}(\omega)} - \tilde{\psi}(\omega) - 1] + e^{-B}\tilde{\psi}(\omega),$$

where the term in brackets decays like $\tilde{\psi}(\omega)^2/2$. Substituting this into (3), we find that

$$\mu(y,\infty) = e^{-B}[G^c(y) + \eta(y)], \tag{23}$$

where

$$G^c(y) := \lim_{L\to\infty} \sum_{n=-\infty}^{\infty} b_n[e^{\tilde{\psi}(n\pi/L)} - \tilde{\psi}(n\pi/L) - 1]e^{-jn\pi y/L},$$

and

$$\eta(y) := \lim_{L\to\infty} \sum_{n=-\infty}^{\infty} b_n\tilde{\psi}(n\pi/L)e^{-jn\pi y/L}.$$

From (22) and (23) it follows that

$$\mathsf{P}(Y > y) = \begin{cases} e^{-B}[G^c(y) + \eta(y)], & y \ge 0, \\ e^{-B}[G^c(y) + \eta(y) + 1], & y < 0. \end{cases}$$

Fortunately, we can evaluate the limit defining η to obtain

$$\eta(y) = \int_y^{\infty} \gamma(\theta)\, d\theta;$$

this follows by applying (3) to the measure whose density is $\gamma(\theta)$ and by noting that the corresponding characteristic function is given by (19).

We now observe that if G^c is absolutely continuous, then $F(y) = 1 - \mathsf{P}(Y > y)$ has density

$$f(y) = e^{-B}\left[\delta(y) + \gamma(y) - \frac{d}{dy}G^c(y)\right]. \tag{24}$$

Remark: Equation (24) generalizes to

$$f(y) = e^{-B}\left[\delta(y) + \gamma(y) + \sum_{i=1}^{m-1}\gamma^{*i}(y) - \frac{d}{dy}G^c_m(y)\right],$$

where γ^{*i} is the convolution of γ with itself i times, and

$$G_m^c(y) := \lim_{L\to\infty} \sum_{n=-\infty}^{\infty} b_n [e^{\tilde{\psi}(n\pi/L)} - \sum_{i=0}^{m} \frac{\tilde{\psi}(n\pi/L)^i}{i!}] e^{-jn\pi y/L}.$$

To approximate the last term in (24), we fit a cubic spline to

$$G_{L,N}^c(y) := \sum_{n=-N}^{N} b_n [e^{\tilde{\psi}(n\pi/L)} - \tilde{\psi}(n\pi/L) - 1] e^{-jn\pi y/L} \tag{25}$$

and then differentiate. Note that even if $\tilde{\psi}(\omega)$ decays like $1/\sqrt{\omega}$, as it does in Example 2 if we assume $A_\nu = 1$ with probability 1, the coefficients in (25) decay like $1/n^2$, and so as $N \to \infty$, $G_{L,N}^c$ converges to a continuous function. Thus, for finite N, (25) will not exhibit Gibbs phenomenon. Of course, if we were to differentiate (25), the new series coefficients would decay like $1/n$, and Gibbs phenomenon would appear. By fitting a spline with appropriately chosen knots to $G_{L,N}^c$ and *then* differentiating, we can avoid the Gibbs oscillations.

A. Error Bounds

By taking L finite in the first sum in (23) we see that there are two sources of error. The first one can be bounded by looking at the measure corresponding to $\varphi(\omega) - e^B$, i.e., to μ itself as was done in Section II. The second source can be bounded by looking at the measure corresponding to $\tilde{\psi}(\omega)$, i.e., to the measure whose density is γ. Now, if $|g(\tau)|$ is bounded by some constant, say $g_{\max}$, then $\gamma_0(\theta) = 0$ for $|\theta| > g_{\max}$. Hence, if A_ν is a bounded random variable, say $A_\nu \le \alpha$, then by (18), $\gamma(\theta) = 0$ for $|\theta| > \alpha g_{\max}$. So, if $L + L_2 > \alpha g_{\max}$, and if $-L + L_1 < -\alpha g_{\max}$, there will be no error due to the second source. This will be the case in the following examples.

B. Numerical Examples

In the following examples we take $A_\nu \equiv 1$ so that $\psi = \psi_0$, $k = k_0$, and $\gamma = \gamma_0$.

Example 2 Continued: Let $a = b = 2$ and $z = 1/2$. Then $g(\tau) = q(\|\tau\|)$, where q is the quadratic spline shown in Fig. 4. We first consider the error bounds. Since $g(\tau) \ge 0$, $\mu(-\infty, 0) = 0$, and so the upper bound in (4) is zero. We now take $L = 7, L_1 = 0$, and $L_2 = 5$. To analyze the lower bound in (5), write

$$\begin{aligned} \mu(L + L_1, \infty) &= \mu(L, \infty) \\ &= \mathsf{P}(Y > L, Y \neq 0) \\ &= \mathsf{P}(Y > L), \quad \text{since } L > 0, \\ &\le e^{-sL} m(s) \\ &= e^{-[sL - k_0(s)]}. \end{aligned}$$

It is easily verified numerically that the maximum value of $7s - k_0(s)$ is greater than 12 and occurs at $s \approx 3$. This yields the bound $e^{-12} \le 6.2 \times 10^{-6}$. Turning now to (25), we take $N = 256$. An FFT provided 93 samples of $G_{L,N}^c$ in the interval $[0, 5.01]$. We then fit a spline with 64 uniformly spaced knots in the proper subinterval $[.01, 5]$. A plot of minus the derivative of this spline is shown in Fig. 5. There appear to be corners at $1/2$ and 1. To confirm this, we plotted minus the second derivative in Fig. 6. Clearly, there is a large downward jump at 1 but only a corner at $1/2$. In addition, although it is difficult to see, there is a jump at 2; just as there are oscillations on both sides of the jump at 1, there are telltale oscillations on both sides of 2. If we add a single knot at .5 and double knots at 1 and 2, and recompute the curve in Fig. 6, we obtain the curve in Fig. 7. Notice how the oscillations around 1 and 2 have disappeared. Based on these observations, we selected a new set of 21 nonuniformly spaced knots, including a single knot at .5 and double knots at 1 and 2. Using the original 93 samples of $G_{L,N}^c$, we fitted a new spline with the set of 21 knots. After differentiating and negating, we obtained the curve shown in Fig. 8. If we add $\gamma(y)$ to this curve and multiply by e^{-B} (cf. (24)), we obtain the graph in Fig. 9. For comparison, we have also plotted with a dashed line the 256-point FFT approximation of $f(y)$ obtained via equation (8). The impulse at the origin is not shown in the figures.

Example 3 Continued: Proceeding as in the previous example, again with $L = 7, L_1 = 0$, and $L_2 = 5$, we obtain the error bound $e^{-8.65} \le 1.8 \times 10^{-4}$. With $N = 256$ there were 93 samples of $G_{L,N}^c$ in the interval $[0, 5.01]$. We started by fitting a spline with 64 uniformly spaced knots in the subinterval $[.01, 5]$. Upon inspection of the result (not shown), we added double knots at 1 and 2 for a total of 68 knots. A plot of minus the derivative of this spline is shown in Fig. 10. Notice that this curve is continuous with corners at 1 and 2. Based on this curve, we selected 17 nonuniformly spaced knots, including double knots at 1 and 2. Using the original 93 samples and the new set of 17 knots, we generated a new spline approximation of $G_{L,N}^c$. A plot of e^{-B} times the quantity $\gamma(y)$ minus the derivative of the 17-knot spline is shown in Fig. 11. For comparison, we have again plotted with a dashed line the corresponding 256-point FFT approximation of $f(y)$.

Example 4 Continued: Let $\overline{N}_r = 2$. With $L = 8, L_1 = 0$, and $L_2 = 6$, we obtained an error bound of $e^{-9.5} \le 7.5 \times 10^{-5}$. With $N = 256$, there were 98 samples of $G_{L,N}^c$ in the interval $[0, 6.06]$. We started by fitting a spline with 50 uniformly spaced knots in the subinterval $[.01, 6]$. Upon observing the result (not shown), we generated a set of 16 nonuniformly spaced knots, including a double knot at 2. A plot of e^{-B} times the quantity $\gamma(y)$ minus the derivative of the 16-knot spline is shown in Fig. 12. The jump at 1

is due to the fact that γ is not continuous there. Again, for comparison, we have plotted with a dashed line the corresponding 256-point FFT approximation of $f(y)$. In [17, Fig. 4(a)], Morris obtained a similar solid curve using an 8192-point FFT.

VI. Approximation of γ_0 and ψ_0

As we have seen, in order to employ the methods of the previous section, we must be able to compute both γ and $\tilde{\psi}$, which can be expressed in terms of γ_0 and ψ_0, respectively (cf. (18), (19) and (15)). In this section we discuss an approximation scheme to compute both γ_0 and ψ_0.

We first consider the Fourier integral in (17). Without loss of generality, we restrict attention to the integral over the right half line,

$$\int_0^\infty e^{j\omega\theta}\gamma_0(\theta)\,d\theta. \tag{26}$$

While γ_0 is integrable since it is the density of a finite measure, it is still possible to have $\gamma_0(0+) = \infty$ as we saw in Example 2. Thus, it may be difficult to compute this integral numerically. We therefore propose the application of integration by parts. However, before doing so, it is convenient to introduce the following notation. For $\theta \geq 0$, let $H_0(\theta) := \Gamma_0(\theta, \infty)$. (For $\theta < 0$ we would use $H_0(\theta) = \Gamma_0(-\infty, \theta)$.) Since Γ_0 is a finite measure with density γ_0, it immediately follows that for $\theta \geq 0$, H_0 is bounded, continuous, nonincreasing, and decays to zero as θ goes to infinity. Also, $H_0'(\theta) = -\gamma_0(\theta)$ for almost every $\theta \geq 0$. Now set

$$H_{n+1}(\theta) \;:=\; -\int_\theta^\infty H_n(\zeta)\,d\zeta, \quad \theta \geq 0. \tag{27}$$

Since H_0 is nonnegative, H_n is always of one sign.

Proposition 3: Let $G_+ := \{\tau : g(\tau) > 0\}$. For $v > 0$, set $S_0(\theta, v) := \mathbf{1}_{[0,v)}(\theta)$, and for $n \geq 1$, set

$$S_n(\theta, v) \;:=\; \begin{cases} (\theta - v)^n/n!, & 0 \leq \theta < v, \\ 0, & \theta > v. \end{cases}$$

Then

$$H_n(\theta) \;=\; \int_{G_+} \lambda(\tau) S_n(\theta, g(\tau))\,d\tau, \quad \theta \geq 0. \tag{28}$$

Furthermore, $H_n(\theta)$ is finite for $\theta \geq 0$ if and only if $H_n(0)$ is finite; i.e., if and only if

$$\int_{G_+} \lambda(\tau) g(\tau)^n\,d\tau \;<\; \infty.$$

If $H_n(0)$ is finite, then H_n is continuous, $H_n(\theta) \to 0$ as $\theta \to \infty$, and $H_n'(\theta) = H_{n-1}(\theta)$ is integrable.

Proof: Use induction and Tonelli's Theorem. □

Remarks: (*i*) Since we are assuming that Γ_0 is a finite measure, $\int_{G_+} \lambda(\tau)\,d\tau < \infty$. Thus, if g is bounded, $H_n(0)$ is finite for all n.

(*ii*) For fixed θ, $S_0(\theta, v)$ is a discontinuous function of v, $S_1(\theta, v)$ is a continuous function of v, and for $n \geq 2$, $S_n(\theta, v)$ is an $n-1$ times continuously differentiable function of v.

Corollary 4: If $H_{n+1}(0)$ is finite, then the integral in (26) is equal to

$$\sum_{k=0}^{n} (-j\omega)^k H_k(0) - (-j\omega)^{n+1} \int_0^\infty e^{j\omega\theta} H_n(\theta)\,d\theta, \tag{29}$$

where H_n is n times continuously differentiable.

Rather than compute the integral (26) numerically, it may be advantageous to use (29) instead, since H_n is smoother than γ_0. We propose using H_1 in the following approximation scheme for computing both γ_0 and ψ_0. Assume $H_2(0)$ is finite. We compute $H_1(\theta)$ at just enough points θ so that we can obtain a good cubic spline approximation of H_1. Denote this approximation by $\tilde{H}_1$. Then $-\tilde{H}_1''$ will provide a piecewise linear approximation of γ_0. Of course, the approximation will break down near the origin if $\gamma_0(0+) = \infty$. To approximate $\psi_0(\omega)$ at $\omega = n\pi/L$ for $n = 0, \ldots, N-1$, we use (29) as follows. Set $\Delta\theta = 2L/M$ for some $M \geq N$. Let $\widehat{H}_1$ denote the piecewise linear approximation of $\tilde{H}_1$ satisfying $\widehat{H}_1(k\,\Delta\theta) = \tilde{H}_1(k\,\Delta\theta)$. Then if we substitute $\widehat{H}_1$ into (29) and use integration by parts, we obtain

$$H_0(0) + \sum_{k=0}^{M-1} c_k [e^{j\omega(k+1)\Delta\theta} - e^{j\omega k\Delta\theta}],$$

where c_k is the slope of $\widehat{H}_1$ on the interval $(k\,\Delta\theta, (k+1)\Delta\theta)$. Assuming M is large enough that $\tilde{H}_1((M-1)\Delta\theta) = \tilde{H}_1(M\,\Delta\theta) = 0$, we will have $c_{M-1} = 0$. Then since $\Delta\theta = 2L/M$, if $\omega = n\pi/L$, the summation can be evaluated for $n = 0, \ldots, M-1$ with a pair of M-point FFTs.

In the continuation of Example 2, we computed ψ_0 exactly using (21). We have also repeated those calculations replacing the true values of $\psi_0(n\pi/L)$ by the just-described approximation. The results were graphically indistinguishable from the solid curves already shown. The particulars were as follows. Using 64 uniformly spaced knots on the subinterval $[10^{-7}, 1 - 10^{-7}]$, we fitted a cubic spline to 96 uniformly spaced samples of H_1 from the interval $[0, 1]$. To approximate ψ_0 we used $M = 1024$. Similar results were also obtained using only 46 nonuniformly spaced samples of H_1 and 16 nonuniformly spaced knots. (Of course, on a fine enough scale, the approximations of γ_0 break down near the origin. For these particular approximations, the breakdown is not visible graphically for $\theta \geq .05$.)

Appendix
Proof of Lemma 1

Our approach is a generalization of [1]. First note that $\mu(0,\infty) = \int_{-\infty}^{\infty} \mathbf{1}_{(0,\infty)}(x)\, d\mu(x)$. Now, for any $L > 0$, define the periodic pulse train β_L with period $2L$ by specifying its values on $(-L, L]$ to be

$$\beta_L(x) = \begin{cases} 1, & 0 < x \le L, \\ 0, & -L < x \le 0. \end{cases}$$

Since $\beta_L(x) \to \mathbf{1}_{(0,\infty)}(x)$ for all x as $L \to \infty$, the dominated convergence theorem implies

$$\int_{-\infty}^{\infty} \beta_L(x)\, d\mu(x) \to \mu(0,\infty). \tag{30}$$

Below we show that $\int_{-\infty}^{\infty} \beta_L(x)\, d\mu(x)$ is given by the infinite series in (1). Before doing so, we first give another proof of (30) that yields a bound on the error when L is finite. Observe that for all x,

$$\beta_L(x) \le \mathbf{1}_{(0,\infty)}(x) + \mathbf{1}_{(-\infty,-L]}(x)$$

and

$$\beta_L(x) \ge \mathbf{1}_{(0,L]}(x) = \mathbf{1}_{(0,\infty)}(x) - \mathbf{1}_{(L,\infty)}(x).$$

Hence, the difference,

$$\int_{-\infty}^{\infty} \beta_L(x)\, d\mu(x) - \mu(0,\infty),$$

is upper bounded by $\mu(-\infty,-L]$ and lower bounded by $-\mu(L,\infty)$. Since μ is a finite measure, $\mu(L,\infty)$ and $\mu(-\infty,-L]$ both converge to zero as L goes to infinity.

The next step in the proof is to approximate β_L by its Fourier series. Set

$$\beta_{L,N}(x) := \sum_{n=-N}^{N} b_n e^{j n \pi x / L},$$

where the b_n are given by (2). Now, as $N \to \infty$, $\beta_{L,N}(x) \to \beta_L(x)$ for all x except at points of discontinuity of β_L. Since μ is finite and nonatomic, and since $\beta_{L,N}(x)$ is uniformly bounded in N and x (with L fixed) [8, p. 151, eq. (10.1.5)], the dominated convergence theorem yields

$$\begin{aligned} \int_{-\infty}^{\infty} \beta_L(x)\, d\mu(x) &= \sum_{n=-\infty}^{\infty} b_n \int_{-\infty}^{\infty} e^{j n \pi x / L}\, d\mu(x) \\ &= \sum_{n=-\infty}^{\infty} b_n \Phi(n\pi/L). \end{aligned}$$

□

References

[1] N. C. Beaulieu, "A simple series for personal computer computation of the error function $Q(\cdot)$," *IEEE Trans. Commun.*, vol. 37, no. 9, pp. 989-991, Sept. 1989.

[2] G. Birkhoff and C. de Boor, "Error bounds for spline interpolation," *J. Math. and Mech.*, vol. 13, pp. 827-835, 1964.

[3] C. de Boor, *A Practical Guide to Splines*. New York: Springer-Verlag, 1978.

[4] P. Billingsley, *Probability and Measure*, 2nd ed. New York: Wiley, 1986.

[5] J. A. Bucklew, *Large Deviations in Decision, Simulation, and Estimation*. New York: Wiley, 1990.

[6] N. Campbell, "The study of discontinuous phenomena," *Proc. Cambr. Phil. Soc.*, vol. 15, pp. 117-136, 1909.

[7] ——, "Discontinuities in light emission," *Proc. Cambr. Phil. Soc.*, vol. 15, pp. 117-136, 1909.

[8] R. E. Edwards, *Fourier Series, A Modern Introduction, Volume 1*. New York: Holt, Rinehart and Winston, 1967.

[9] G. J. Foschini, R. D. Gitlin, and J. Salz, "Optimum direct detection for digital fiber optic communication systems," *Bell Syst. Tech. J.*, vol. 54, no. 8, pp. 1389-1430, Oct. 1975.

[10] E. N. Gilbert and H. O. Pollak, "Amplitude distribution of shot noise," *Bell Syst. Tech. J.*, vol. 39, pp. 333-350, Mar. 1960.

[11] R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed. New York: McGraw-Hill, 1973.

[12] C. W. Helstrom and C.-L. Ho, "Analysis of avalanche diode receivers by saddlepoint integration," *IEEE Trans. Commun.*, submitted.

[13] A. O. Hero, "Timing estimation for a filtered Poisson process in Gaussian noise," *IEEE Trans. Inform. Theory*, vol. 37, no. 1, pp. 92-106, Jan. 1991.

[14] T. T. Kadota, "Approximately optimum detection of deterministic signals in Gaussian and compound Poisson noise," *IEEE Trans. Inform. Theory*, vol. 34, no. 6, pp. 1517-1527, Nov. 1988.

[15] S. B. Lowen and M. C. Teich, "Power-law shot noise," *IEEE Trans. Inform. Theory*, vol. 36, no. 6, pp. 1302-1318, Nov. 1990.

[16] J. E. Mazo and J. Salz, "On optical data communication via direct detection of light pulses," *Bell Syst. Tech. J.*, vol. 55, no. 3, pp. 347-369, Mar. 1976.

[17] G. M. Morris, "Scene matching using photon-limited images," *J. Opt. Soc. Am. A*, vol. 1, no. 5, pp. 482-488, May 1984.

[18] A. Papoulis, "High density shot noise and Gaussianity," *J. Appl. Prob.*, vol. 8, no. 1, 118-127, Mar. 1971.

[19] S. O. Rice, "Mathematical analysis of random noise," *Bell Syst. Tech. J.*, vol. 23, pp. 282-332, July 1944; vol. 24, pp. 46-156, Jan. 1945. [Reprinted in *Selected Papers on Noise and Stochastic Processes*, N. Wax, Ed. New York: Dover, 1954, pp. 133-294].

[20] W. J. Richter Jr. and T. I. Smits, "Numerical evaluation of Rice's integral representation of the probability density function for Poisson impulse noise," *J. Acoust. Soc. Amer.*, vol. 56, pp. 481-496, 1974.

[21] W. Schottky, "Über spontane Stromschwankungen in verschiedenen Elektrizitätsleitern," *Annalen der Physik*, vol. 57, pp. 541-567, 1918.

[22] D. L. Snyder, *Random Point Processes*. New York: Wiley, 1975.

[23] O.-C. Yue, R. Lugannani, and S. O. Rice, "Series approximations for the amplitude distribution and density of shot processes," *IEEE Trans. Commun.* vol. COM-26, no. 1, pp. 45-54, Jan. 1978.

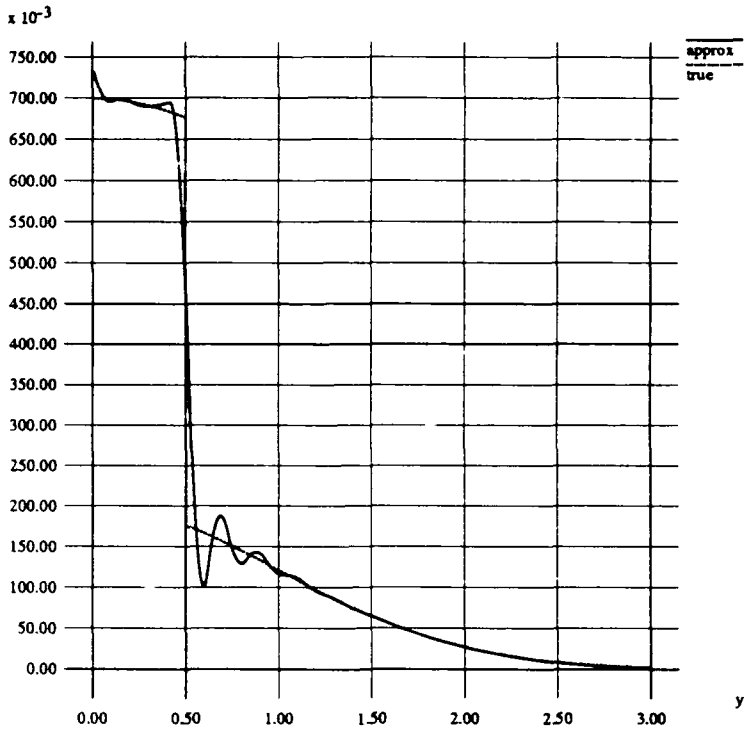


Fig. 1. Approximation of $f(y)$ based on 30-knot spline in Example 1.

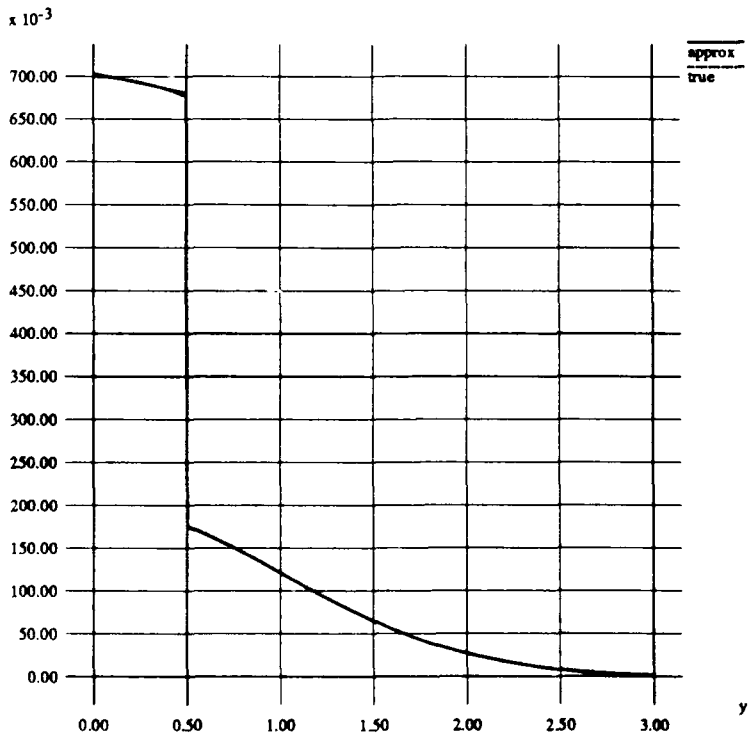


Fig. 2. Approximation of $f(y)$ based on 8-knot spline in Example 1.

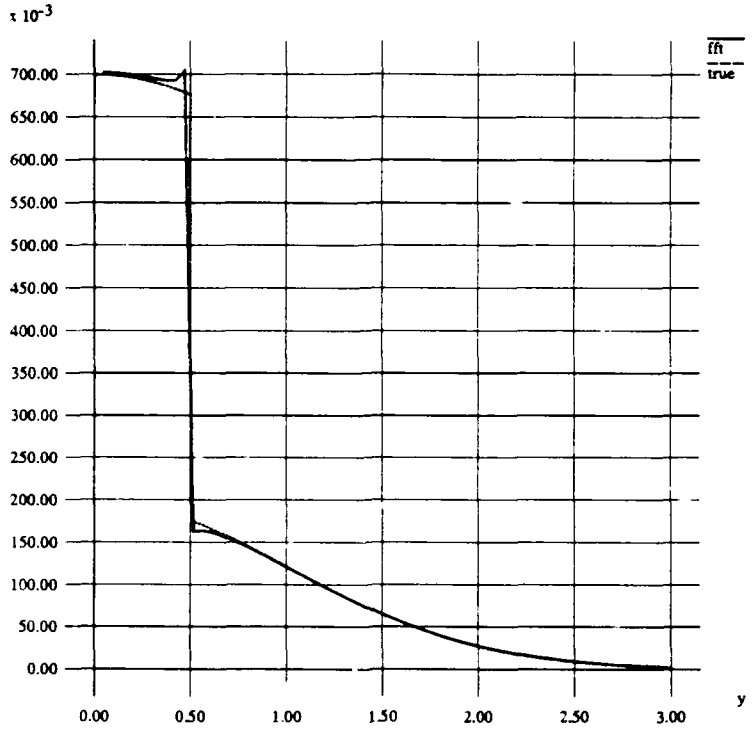


Fig. 3. 256-point FFT approximation of $f(y)$ in Example 1.

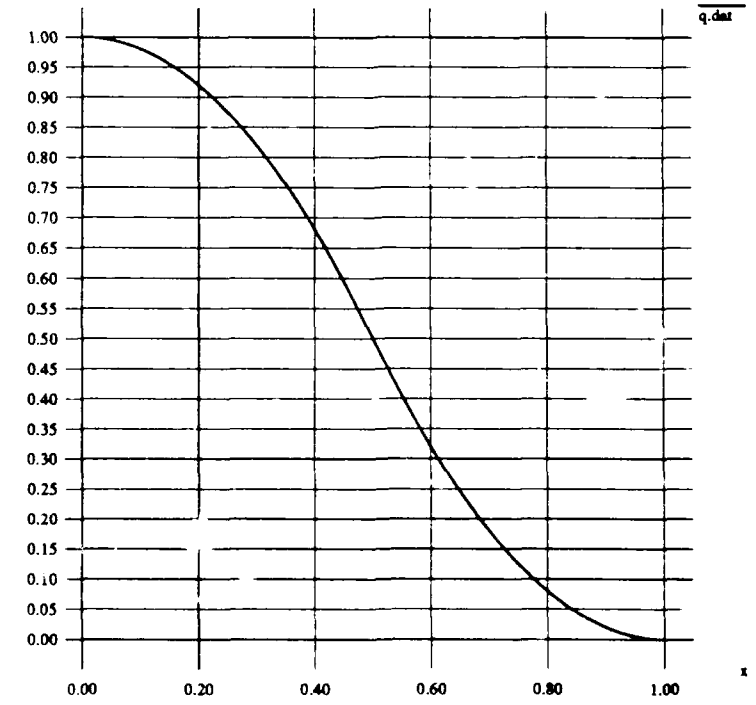


Fig. 4. Plot of $q(x)$ in Example 2 with $a = b = 2$ and $r = 1/2$.

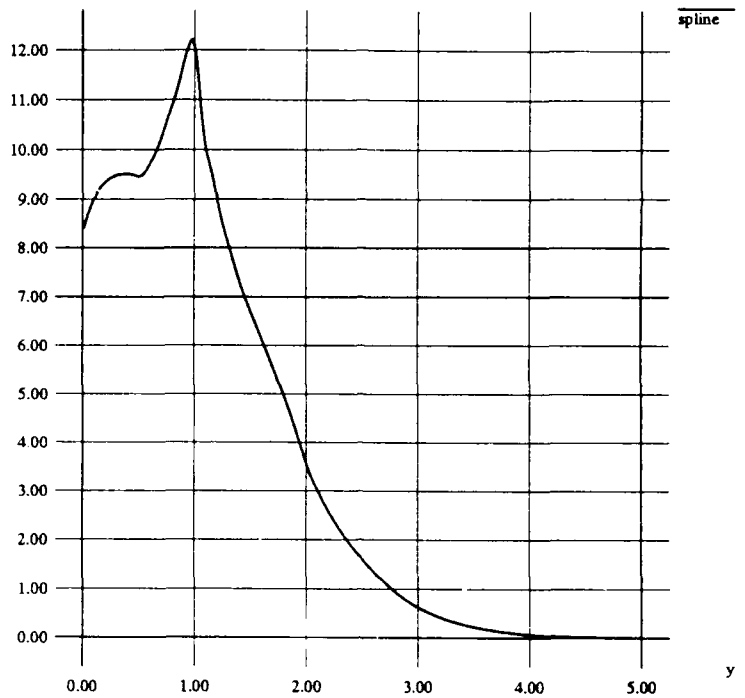


Fig. 5. $-\frac{d}{dy}$ of 64-knot spline for $G^{c}_{L,N}(y)$ in Example 2 Continued.

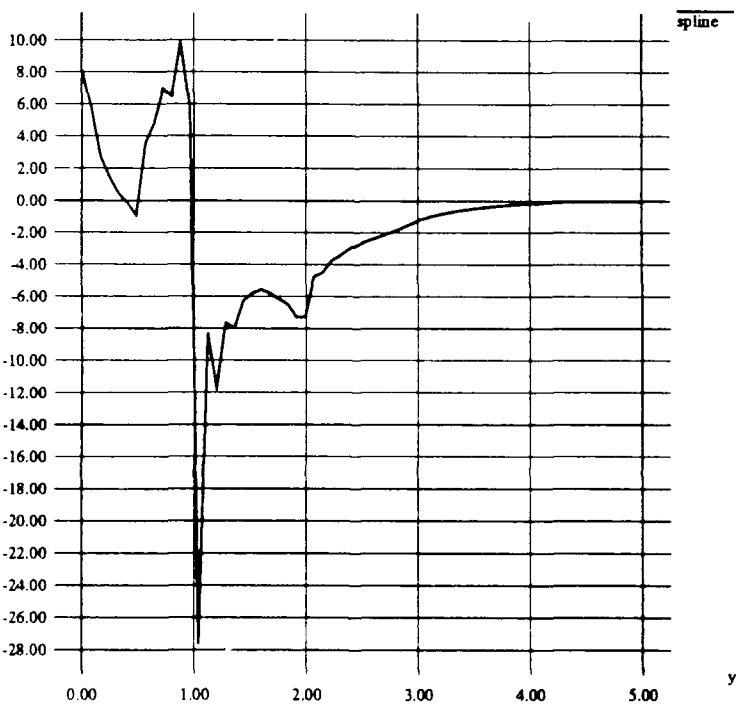


Fig. 6. $-\frac{d^2}{dy^2}$ of 64-knot spline for $G^{c}_{L,N}(y)$ in Example 2 Continued.

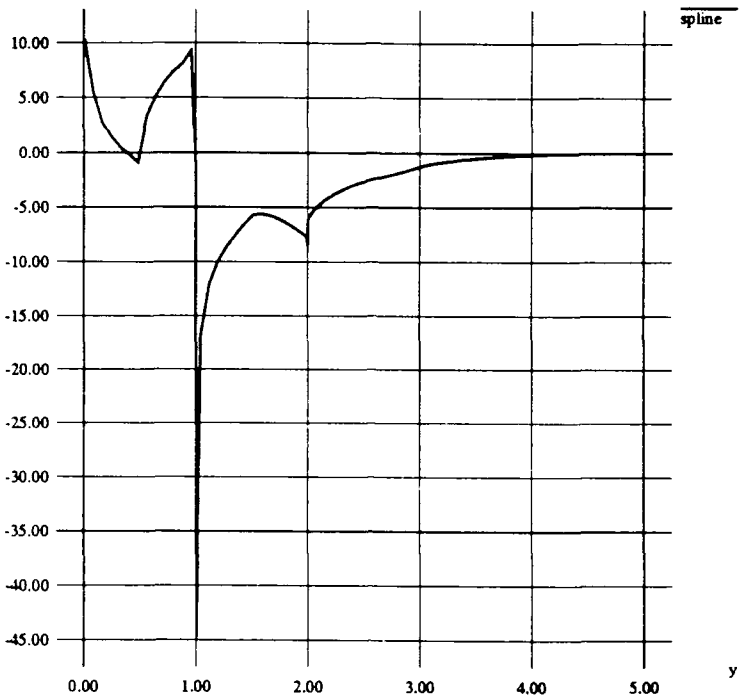


Fig. 7. $-\frac{d^2}{dy^2}$ of 69-knot spline for $G^c_{L,N}(y)$ in Example 2 Continued.

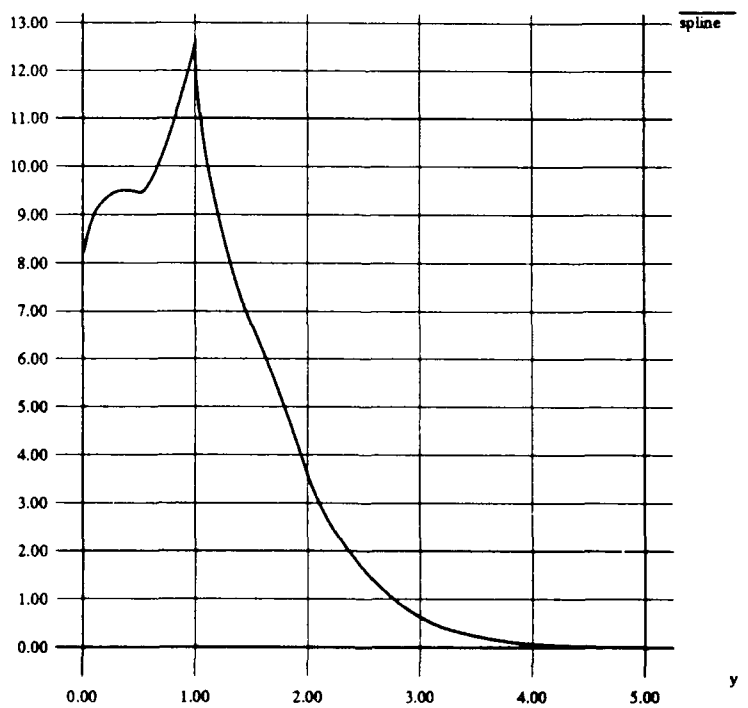


Fig. 8. $-\frac{d}{dy}$ of 21-knot spline for $G^c_{L,N}(y)$ in Example 2 Continued.

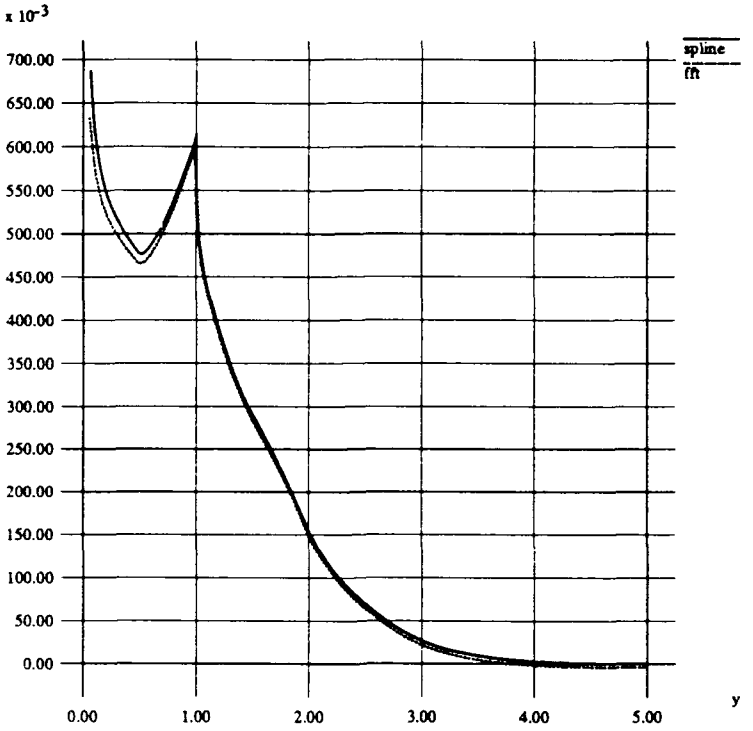


Fig. 9. Approximation of $f(y)$ based on 21-knot spline in Example 2 Continued.

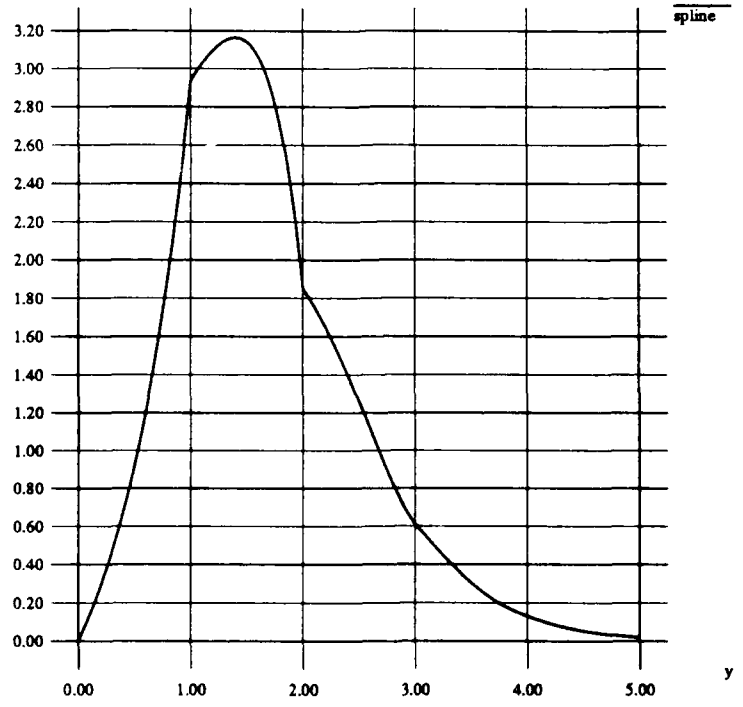


Fig. 10. $-\frac{d}{dy}$ of 68-knot spline for $G^{c}_{L,N}(y)$ in Example 3 Continued.

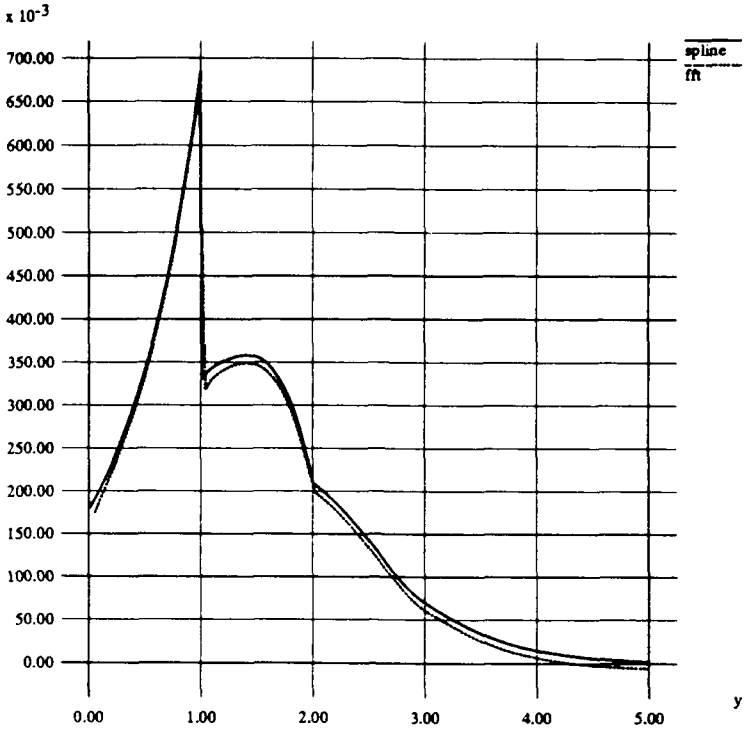


Fig. 11. Approximation of $f(y)$ based on 17-knot spline in Example 3 Continued.

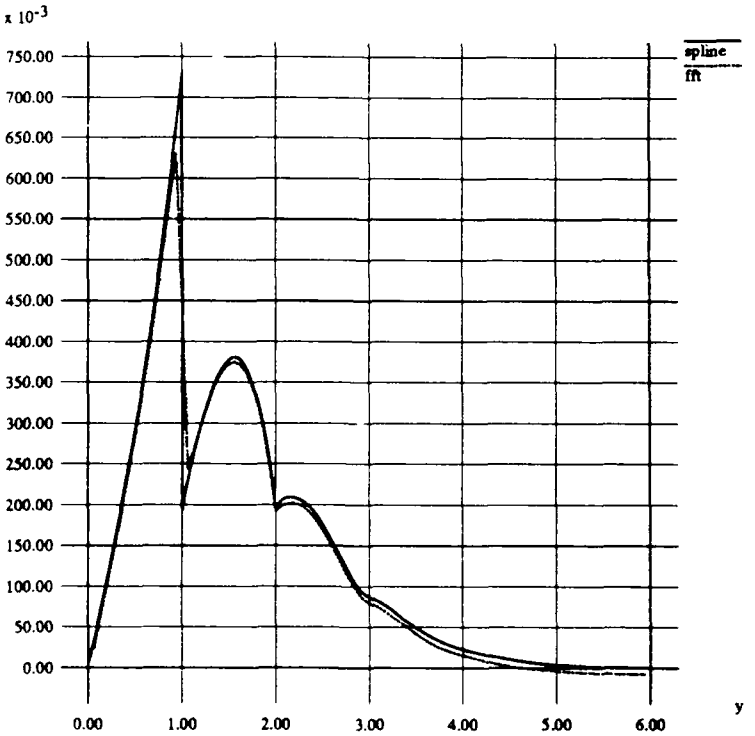


Fig. 12. Approximation of $f(y)$ based on 16-knot spline in Example 4 Continued.

Appendix D

Reprint of Reference [7]

To appear in IEEE TRANSACTIONS ON IMAGE PROCESSING

Image Detection under Low-Level Illumination

Raúl E. Sequeira, *Student Member, IEEE,* John A. Gubner, *Member, IEEE,*
and Bahaa E. A. Saleh, *Fellow, IEEE*

Abstract — Image detection under low-light-level conditions is treated as a hypothesis-testing problem in which the observations are modeled as a shot-noise process. Since computing the likelihood ratio for shot-noise processes is not feasible, we propose the use of a one-dimensional test statistic obtained by filtering and sampling the observations. The filter is chosen to maximize a generalized signal-to-noise ratio. The likelihood ratio for the one-dimensional test statistic is evaluated numerically by inverting the corresponding characteristic function under each hypothesis.

Index Terms – Hypothesis testing, generalized signal-to-noise ratio, shot noise, filtered point process.

This research was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-90 0181 and in part by the University of Wisconsin-Madison Graduate School. This work was presented in part at the Optical Society of America Annual Meeting, San Jose, CA, Nov. 1991.

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.

I. Introduction

When images are produced under high-level illumination, they are often modeled as the sum of a "signal" image plus signal-independent Gaussian noise. However, when an image is formed under low-light-level illumination, it can be better modeled as a filtered point process, also known as shot noise, described as follows.

The process of image acquisition consists of the measurement of arriving photons in the image plane. In practical situations it is difficult to measure the exact location of the photons since these are filtered by the finite response of the imaging device. What is measured, instead, is the superposition of the responses of the imaging system to each arriving photon. This superposition, measured at a point $x \in \mathbb{R}^2$, can be described by the random variable

$$Z(x) = \sum_{\nu} h(x - X_\nu), \tag{1.1}$$

where $\{X_\nu\}$ denotes the set of positions at which photons are detected, and h represents the impulse response or point spread function of the imaging device. A block diagram of the imaging system is shown in Fig. 1.

We consider the following hypothesis-testing problem. Under hypothesis H_i, $i = 0, 1$, the $\{X_\nu\}$ are points of a two-dimensional Poisson process with nonnegative intensity $\lambda_i(x)$, $x \in \mathbb{R}^2$.

Special cases of this problem that have previously been addressed include:

1. The exact locations of the photons are available. This is equivalent to the point spread function $h(x)$ being equal to a Dirac delta function $\delta(x)$, in which case one has a detection problem with Poisson-process observations. When one has Poisson-process observations, the likelihood ratio (LR) is of course well known [9, p. 94]. In [4], a suboptimal detection scheme that could be easily implemented was considered. This led to the consideration of a correlation detector in which the Poisson-process observations were passed through a linear filter (taken to be one of the intensity functions) and sampled. This led to a hypothesis test based on a single shot-noise random variable.

2. Counts of photons in disjoint regions are available, and hence one is faced with a Poisson counting process detection problem. This case can be regarded as a filtered Poisson process with special form of h, followed by sampling. The LR is well known for this special case also [9, p. 94]. In [10], a correlation scheme was used for classification. The Poisson counting process observations (with counts being either 0 or 1 with high probability) were cross-correlated with various reference functions. Three reference functions were considered, one of which was constructed so that the value of the correlation between that function and the observed image approximates the value of the logarithm of the LR.

Little work has been done for the more general case of filtered point-process observations due to the difficulty of computing the density and distribution functions involved [3].

II. Hypothesis Testing with Shot-Noise Observations

A. *Preliminary Considerations*

Using the mathematical model described in Section I, we would like to decide whether λ_0 or λ_1 is the true intensity of the underlying Poisson process that gives rise to our observations $\{Z(x)\}$. Clearly, a likelihood ratio test (LRT) is called for [7, p. 11]. Unfortunately, a formula for the LR of the continuum of observations $\{Z(x)\}$ does not seem to be available in the literature. In fact, even

if we based our decision in a finite number of samples of $\{Z(x)\}$, say $Z(x_1),\ldots,Z(x_K)$, the LR would be obtained by inverting the K-variate characteristic function of $Z(x_1),\ldots,Z(x_K)$ under each hypothesis to obtain the K-variate density under each hypothesis. The quotient of these densities would yield the LR. This approach is impractical unless K is very small. It is because of these considerations that we introduce the following suboptimal approach.

B. A Suboptimal Detector

As a suboptimal detector scheme, we propose that the received image $\{Z(x)\}$ be passed through a linear filter g (to be chosen later in Section II-E) and then sampled, as shown in Fig. 2. The final discrimination of hypotheses will be based on the sample T. More precisely, let

$$Y(x) \triangleq \int g(x-u)Z(u)du, \tag{2.1}$$

where $Z(\cdot)$ is given by (1.1). (Here and in the sequel integrals are understood as being over all of $\mathbb{R}^2$ unless otherwise indicated. One-dimensional integrals over $\mathbb{R}$ are indicated by $\int_{-\infty}^{\infty}$). Observe that if we set

$$\tilde{g}(x) \triangleq \int g(x-u)h(u)du, \tag{2.2}$$

then

$$Y(x) = \sum_{\nu} \tilde{g}(x-X_\nu). \tag{2.3}$$

Clearly (2.3) has the same form as (1.1). In other words, when the shot-noise process $\{Z(x)\}$ is passed through the linear system g, the output $\{Y(x)\}$ is also a shot-noise process. The final processing step shown in Fig. 2 is sampling. We set

$$T \triangleq Y(0) = \sum_{\nu} \tilde{g}(-X_\nu), \tag{2.4}$$

and perform an LRT based on T.

C. The Likelihood Ratio for T

Let $F_i(t) \triangleq \mathcal{P}_i(T \le t)$, $i = 0, 1$, be the cumulative probability distribution of T given that λ_i is the true intensity of the underlying Poisson process that models the location at which arriving photons strike the imaging system. Clearly, if no photons arrive, $T = 0$. The probability of this event is

$$\mathcal{P}_i(T=0) = e^{-\Lambda_i},$$

where

$$\Lambda_i \triangleq \int \lambda_i(x)\,dx.$$

In our applications we have $\Lambda_i < \infty$. We thus expect $F_i(t)$ to have the form

$$F_i(t) = \begin{cases} \displaystyle\int_{-\infty}^{t} f_i(\tau)\,d\tau, & t < 0, \\ \displaystyle e^{-\Lambda_i} + \int_{-\infty}^{t} f_i(\tau)\,d\tau, & t \ge 0, \end{cases} \tag{2.5}$$

for some nonnegative f_i satisfying

$$\int_{-\infty}^{\infty} f_i(\tau)\,d\tau = 1 - e^{-\Lambda_i}.$$

Remark: In general $F_i(t)$ could have jumps at points $t \neq 0$, depending on the behavior of $\tilde{g}$. In our examples, this does not occur.

Assuming equally likely hypotheses, if we observe $T = t$, we perform the LRT

$$\begin{cases} \Lambda_0 - \Lambda_1 \underset{H_0}{\overset{H_1}{\gtrless}} 0, & t = 0, \\ \dfrac{f_1(t)}{f_0(t)} \underset{H_0}{\overset{H_1}{\gtrless}} 1, & t \neq 0. \end{cases} \tag{2.6}$$

The numerical calculation of f_0 and f_1 is discussed in Appendix A.

D. The Probability of Error

The probability of error incurred using the LRT (2.6) is denoted by P_e; we can write an expression for P_e as follows. First, let D_0 denote the set of t such that we decide in favor of H_0. Clearly, $0 \in D_0$ if and only if $\Lambda_0 < \Lambda_1$. For $t \neq 0$, $t \in D_0$ if and only if $f_0(t) < f_1(t)$. Let D_1 denote the complement of D_0. We write $D_1 = D_0^c$. Then, under equally likely hypotheses,

$$\begin{aligned} P_e &= \frac{1}{2}\Big\{\wp_1[T \in D_0] + \wp_0[T \in D_1]\Big\} \\ &= \frac{1}{2}\Big\{1 + \int_{D_0} [dF_1(t) - dF_0(t)]\Big\}. \end{aligned} \tag{2.7}$$

Let $I_A(t)$ denote the indicator function of a set A. In other words, $I_A(t) = 1$ if $t \in A$ and $I_A(t) = 0$ otherwise. Using (2.7) and (2.5) we can then write

$$\begin{aligned} P_e &= \frac{1}{2}\Big\{1 + [e^{-\Lambda_1} - e^{-\Lambda_0}] I_{D_0}(0) + \int_{D_0 \cap \{0\}^c} (f_1(t) - f_0(t))\, dt\Big\} \\ &= \frac{1}{2}\Big\{1 + [e^{-\Lambda_1} - e^{-\Lambda_0}] I_{D_0}(0) + \int_{D_0} (f_1(t) - f_0(t))\, dt\Big\}. \end{aligned} \tag{2.8}$$

In our applications, D_0 will turn out to be an interval, or a union of disjoint intervals. Hence the last term in (2.8) can easily be computed if we have a simple way to evaluate

$$G_i(t) \triangleq \int_{-\infty}^{t} f_i(\tau)\, d\tau. \tag{2.9}$$

We discuss this further in Appendix B.

E. Selecting the Filter g

Ideally we would like to select g to minimize the probability of a decision error, P_e. Since the dependence of P_e on g is not readily apparent, we introduce the following ad-hoc criterion for selecting g. We would like to choose g to maximize the generalized "signal-to-noise ratio" (cf. [1, eq. (18)], [2, eq. (6)])

$$\frac{(\mu_1 - \mu_0)^2}{\sigma_1^2 + \sigma_0^2}, \tag{2.10}$$

where μ_i and σ_i^2 are the mean and variance of T under hypothesis $i = 0, 1$. Since Y in (2.3) is a shot-noise process, it follows from (2.4) and [5, pp. 382-383] that

$$\begin{aligned} \mu_i &\triangleq \mathsf{E}_i[T] \\ &= \int \tilde{g}(-x)\lambda_i(x)\, dx, \end{aligned} \tag{2.11}$$

and

$$\begin{aligned}\sigma_i^2 &\triangleq \mathsf{E}_i[(T-\mu_i)^2] \\ &= \int \tilde{g}(-x)^2 \lambda_i(x)\,dx. \end{aligned} \tag{2.12}$$

Using (2.11) and (2.12), it follows from the Cauchy-Schwarz inequality that $\tilde{g}$ maximizes (2.10) if and only if

$$\tilde{g}(-x) = c\,\frac{\lambda_1(x)-\lambda_0(x)}{\lambda_1(x)+\lambda_0(x)}, \tag{2.13}$$

where c is an arbitrary constant. Using (2.2), (2.13) then becomes

$$\int g(-x-u)h(u)\,du = c\,\frac{\lambda_1(x)-\lambda_0(x)}{\lambda_1(x)+\lambda_0(x)}. \tag{2.14}$$

Unfortunately, (2.14) may not have a solution. For example, if g and h are square integrable, the left-hand side of (2.14) will be a continuous function of x, while the right-hand side need not be, as is the case in our examples in Section III-C. In order to avoid this problem, we a priori constrain g to be of the form

$$g(x) = \sum_{k=1}^{K} w_k \delta(x+x_k), \tag{2.15}$$

where K and the locations $x_1,\ldots,x_K$ are chosen in some heuristic fashion, perhaps as a uniform grid, and the weights are chosen to maximize (2.10). Observe that if (2.15) holds, then it follows from (2.1) that

$$Y(x) = \sum_{k=1}^{K} w_k Z(x+x_k),$$

and thus

$$T \triangleq Y(0) = \sum_{k=1}^{K} w_k Z(x_k),$$

i.e., the test statistic T is a weighted superposition of the measurements $Z(x_1),\ldots,Z(x_K)$. Now, let $\mathbf{w} \triangleq [w_1,\ldots,w_K]'$ and $\mathbf{Z} \triangleq [Z(x_1),\ldots,Z(x_K)]'$. Let $\mathbf{m}_i \triangleq \mathsf{E}_i[\,\mathbf{Z}\,]$ and $\mathbf{\Gamma}_i \triangleq \mathsf{E}_i[(\mathbf{Z}-\mathbf{m}_i)(\mathbf{Z}-\mathbf{m}_i)']$. Then $T = \mathbf{w}'\mathbf{Z}$, and

$$\mu_i = \mathbf{w}'\mathbf{m}_i \quad \text{and} \quad \sigma_i^2 = \mathbf{w}'\mathbf{\Gamma}_i\mathbf{w}, \tag{2.16}$$

where the kth entry of the vector $\mathbf{m}_i$ is (cf. (2.11))

$$\int h(x_k-x)\lambda_i(x)\,dx \tag{2.17}$$

and the $k\ell$ entry of the matrix $\mathbf{\Gamma}_i$ is (cf. (2.12))

$$\int h(x_k-x)h(x_\ell-x)\lambda_i(x)\,dx. \tag{2.18}$$

Letting $\mathbf{m} = \mathbf{m}_1 - \mathbf{m}_0$ and $\mathbf{\Gamma} = \mathbf{\Gamma}_0 + \mathbf{\Gamma}_1$, we see that under the constraint (2.15), (2.10) becomes

$$\frac{|\mathbf{w}'\mathbf{m}|^2}{\mathbf{w}'\mathbf{\Gamma}\mathbf{w}}. \tag{2.19}$$

By the Cauchy-Schwarz inequality, $\mathbf{w}$ maximizes (2.19) if and only if $\mathbf{\Gamma}\mathbf{w} = c\,\mathbf{m}$, where c is an arbitrary constant. For the numerical examples in Section III-C we take $c = 1$, and $\mathbf{\Gamma}\mathbf{w} = \mathbf{m}$ is

easily solved using the NAG routine F04ASF. The NAG routine D01FCF is used to compute (2.17) and (2.18).

Remark: One of the reviewers has suggested selecting g so that

$$\tilde{g}(-x) \;=\; \int g(-x-u)h(u)\,du \;=\; \ln\frac{\lambda_1(x)}{\lambda_0(x)}, \tag{2.20}$$

the idea being that (2.4) would then be equal (up to an additive constant) to the logarithm of the LR of the point process itself. A related idea was used in [10]. Unfortunately, (2.20) may not have a solution for the same reasons given following (2.14).

III. Hypothesis Testing and Error Performance

In this section we apply the preceding ideas to several examples. For comparison, we also discuss the consequences of assuming that T has a Gaussian distribution under each hypothesis; this assumption was used, under somewhat higher light levels, in [4, 10] based on the Central Limit Theorem. Under certain conditions this approximation is adequate [6], and it avoids the burden of computing $f_i(t)$ and $G_i(t)$ by the numerical evaluation of inverse Fourier transforms. However, under the low-light-level conditions considered here, we do not expect the Gaussian approximation to work well, and this is indeed the case.

A. Likelihood Ratio Test

Following the observation $T = t$, the LRT is given by (2.6). In the numerical examples discussed below, we plot $f_0(t)$ and $f_1(t)$ (Figs. 4–6) and see that the equation

$$\frac{f_1(t)}{f_0(t)} \;=\; 1$$

has at most one solution of interest, denoted by η. Hence, if $t \neq 0$, the LRT reduces to the single threshold test

$$t \underset{H_0}{\overset{H_1}{\gtrless}} \eta.$$

B. Gaussian Test (GT)

A simple test to write down is the following. Let

$$p_i(t) \;\triangleq\; \frac{1}{\sqrt{2\pi}\sigma_i}e^{-(t-\mu_i)^2/2\sigma_i^2}, \tag{3.1}$$

where μ_i and σ_i^2 are given by (2.16). In other words, we are assuming that T is normal, but with the correct mean and variance. We consider the test

$$\frac{p_1(t)}{p_0(t)} \underset{H_0}{\overset{H_1}{\gtrless}} 1.$$

In the examples discussed below, $\sigma_1^2 \geq \sigma_0^2$, and the decision regions for this test are easily shown to be

$$D_0 \;=\; \left\{t \in \mathbb{R} : -\gamma - \frac{b}{a} \leq t \leq \gamma - \frac{b}{a}\right\} \qquad \text{and} \qquad D_1 \;=\; D_0^c,$$

where $a \triangleq \sigma_1^2 - \sigma_0^2$, $b \triangleq \mu_1\sigma_0^2 - \mu_0\sigma_1^2$, and

$$\gamma \triangleq \left[\frac{1}{a}\{\sigma_0^2\sigma_1^2 \ln\frac{\sigma_1^2}{\sigma_0^2} + \mu_1^2\sigma_0^2 - \mu_0^2\sigma_1^2\} + \frac{b^2}{a^2}\right]^{\frac{1}{2}}.$$

In the examples discussed below, the integrals $\int_{-\infty}^{-\gamma-b/a} p_i(t)\,dt$ are negligible, and so we consider the following single-threshold test. We set $\eta = \gamma - b/a$ and use $D_0 = (-\infty, \eta)$. Even with this choice of D_0, the probability of error is given by (2.8), which requires complicated numerical integration as discussed in Appendix B. However, we also consider P_{Ge}, the "Gaussian approximation" of P_e, that we define by

$$P_{Ge} \triangleq \frac{1}{2}\left\{\int_{-\infty}^{\eta} p_1(t)\,dt + \int_{\eta}^{\infty} p_0(t)\,dt\right\}.$$

In the examples below, P_{Ge} was computed easily with the NAG routine S15ABF for evaluating the cumulative distribution of the standard normal density.

C. Examples

We compare the error performance of the two tests in three examples. In each example we consider three cases, sampling at $K = 1$, 5, and 10 points. For this purpose, let

$$\begin{aligned}
\mathcal{R} &= \{x = (u,v) : 0 \le u \le 1 \text{ and } 0 \le v \le 1\},\\
\lambda_0(x) &= \begin{cases} c_0 I_0(x) + n, & x \in \mathcal{R},\\ 0, & x \notin \mathcal{R},\end{cases}\\
\lambda_1(x) &= \begin{cases} c_1 I_e(x) + n, & x \in \mathcal{R},\\ 0, & x \notin \mathcal{R},\end{cases}\\
h(x) &= e^{-50(u^2+v^2)},
\end{aligned}$$

where $I_0(x)$ and $I_e(x)$ are the indicator functions of the shaded regions in Fig. 3, and n is a constant background level. The sampling points, $x_1, \ldots, x_{10}$, are shown in Fig. 4 and are explicitly listed in Table I.

TABLE I
THE SAMPLING POINTS

k	x_k	k	x_k
1	(0.5,0.5)	6	(0.275,0.825)
2	(0.7,0.175)	7	(0.5,0.65)
3	(0.725,0.325)	8	(0.725,0.5)
4	(0.725,0.825)	9	(0.5,0.325)
5	(0.275,0.5)	10	(0.5,0.175)

When $K = 1$, we set $w_1 = 1$ and hence $T = Z(x_1)$.

Example 1: In this example $c_0 = c_1 = 100$ and $n = 10$. Then $\Lambda_0 = 43.00$ and $\Lambda_1 = 44.87$. The weights $\{w_k\}$ for the case $K = 5$ are $1.100019, 2.603646\times10^{-1}, -8.424318\times10^{-1}, -1.044526\times10^{-2}, -8.713632\times10^{-2}$, and for $K = 10$, $1.191592, 3.038038\times10^{-1}, -9.131236\times10^{-1}, 4.417913\times10^{-3}, -7.798749\times10^{-2}, 1.564578\times10^{-2}, -2.080950\times10^{-1}, 9.888259\times10^{-2}, -9.350529\times10^{-2}, 4.023416\times10^{-3}$. Table II contains the means and variances of the filtered point process for the three cases. Table III contains the thresholds for the two tests under consideration, the corresponding decision regions D_0 and probabilities of error. The value of P_{Ge} is also included. A plot of f_0 and f_1 for each case is shown in Fig. 5.

TABLE II

PARAMETERS FOR EXAMPLE 1

	$K = 1$	$K = 5$	$K = 10$
μ_0	1.52	−1.27	−1.38
σ_0^2	0.422	1.53	1.61
μ_1	4.40	3.25	3.21
σ_1^2	2.58	2.98	2.99

TABLE III

TEST PERFORMANCE IN EXAMPLE 1

K	Test	η	D_0	P_e	P_{Ge}
1	LRT	2.57	$[0, \eta)$	0.0936	-
	GT	2.65	$[0, \eta)$	0.0949	0.0887
5	LRT	0.720	$(-\infty, \eta)$	0.0523	-
	GT	0.773	$(-\infty, \eta)$	0.0525	0.0628
10	LRT	0.650	$(-\infty, \eta)$	0.0506	-
	GT	0.706	$(-\infty, \eta)$	0.0507	0.0617

From Table III, we see that a reduction of 44.1% in P_e is obtained for the LRT when using $K = 5$ instead of $K = 1$, and a further reduction of 3.25% is obtained by using $K = 10$.

Example 2: In this example $c_0 = c_1 = 50$ and $n = 5$. Then $\Lambda_0 = 21.5$ and $\Lambda_1 = 22.4$. The weights $\{w_k\}$ for the case $K = 5$ are 1.100019, 2.603646×10^{-1}, -8.424318×10^{-1}, -1.044526×10^{-2}, -8.713632×10^{-2}, and for $K = 10$, 1.191592, 3.038038×10^{-1}, -9.131236×10^{-1}, 4.417913×10^{-3}, -7.798749×10^{-2}, 1.564578×10^{-2}, -2.080950×10^{-1}, 9.888259×10^{-2}, -9.350529×10^{-2}, 4.023416×10^{-3}. Table IV contains the means and variances of the filtered point process for the three cases. Table V contains the thresholds for the two tests under consideration, the corresponding decision regions D_0 and probabilities of error. The value of P_{Ge} is also included. A plot of f_0 and f_1 for each case is shown in Fig. 6.

TABLE IV

PARAMETERS FOR EXAMPLE 2

	$K = 1$	$K = 5$	$K = 10$
m_0	0.759	−0.63	−0.691
σ_0^2	0.211	0.760	0.800
m_1	2.20	1.62	1.601
σ_1^2	1.29	1.49	1.50

TABLE V

TEST PERFORMANCE IN EXAMPLE 2

K	Test	η	D_0	P_e	P_{Ge}
1	LRT	1.29	$[0, \eta)$	0.177	-
	GT	1.44	$[0, \eta)$	0.182	0.157
5	LRT	0.389	$(-\infty, \eta)$	0.129	-
	GT	0.464	$(-\infty, \eta)$	0.129	0.138
10	LRT	0.360	$(-\infty, \eta)$	0.126	-
	GT	0.423	$(-\infty, \eta)$	0.127	0.137

From Table V, we see that a reduction of 27.1% in P_e is obtained for the LRT when using $K = 5$ instead of $K = 1$, and a further reduction of 2.32% is obtained by using $K = 10$.

Example 3: In this example $c_0 = c_1 = 16$ and $n = 5$. Then $\Lambda_0 = 10.28$ and $\Lambda_1 = 10.58$. The weights $\{w_k\}$ for the case $K = 5$ are 7.590372×10^{-1}, 1.791687×10^{-1}, -5.971229×10^{-1}, -8.206410×10^{-3}, -5.451747×10^{-2}, and for $K = 10$, 8.767560×10^{-1}, 2.073143×10^{-1}, -6.402233×10^{-1}, 7.654726×10^{-3}, -4.801134×10^{-2}, 1.534052×10^{-2}, -1.978471×10^{-1}, 7.396542×10^{-2}, -1.243330×10^{-1}, 2.736930×10^{-2}. Table VI contains the means and variances of the filtered point process for the three cases. Table VII contains the thresholds for the two tests under consideration, the corresponding decision regions D_0 and probabilities of error. The value of P_{Ge} is also included. A plot of f_0 and f_1 for each case is shown in Fig. 7.

TABLE VI
PARAMETERS FOR EXMAPLE 3

	$K = 1$	$K = 5$	$K = 10$
m_0	0.457	−0.0894	−0.122
σ_0^2	0.174	0.199	0.204
m_1	0.918	0.414	0.398
σ_1^2	0.518	0.305	0.316

TABLE VII
TEST PERFORMANCE IN EXAMPLE 3

K	Test	η	D_0	P_e	P_{Ge}
1	LRT	0.675	$[0, \eta)$	0.335	-
	GT	0.893	$[0, \eta)$	0.348	0.310
5	LRT	0.188	$(-\infty, \eta)$	0.312	-
	GT	0.236	$(-\infty, \eta)$	0.313	0.303
10	LRT	0.172	$(-\infty, \eta)$	0.309	-
	GT	0.212	$(-\infty, \eta)$	0.311	0.300

From Table VII, we see that a reduction of 6.87% in P_e is obtained for the LRT when using $K = 5$ instead of $K = 1$, and a further reduction of 0.962% is obtained by using $K = 10$.

IV. DISCUSSION AND CONCLUSION

We have treated image detection at low light levels as a binary hypothesis-testing problem based on a 1-dimensional test statistic. This statistic was obtained by filtering the received image and then sampling at one point. The filter we used was obtained by maximizing an ad-hoc signal-to-noise ratio. In the examples we considered, we found that the largest weights of the filter (for the cases $K = 5$ and $K = 10$) correspond to the locations where the "o" and the "e" do not overlap. This makes intuitive sense: at x_1 the "e" is present and w_1 is the largest positive weight; at x_3 the "o" is present and w_3 is the largest-magnitude negative weight. Since x_1 and x_3 are the most important points for discrimination between H_0 and H_1, we observed little improvement in peformance when using $K = 10$ instead of $K = 5$. The lower the intensity of the point process, the harder it is to discriminate between the hypotheses. The samples in Example 3 bear less information for discrimination than those in the other two examples. This resulted in a much smaller improvement in performance in Example 3 than in the other two examples when going from $K = 1$ to $K = 5$. We compared our LRT with a test that uses Gaussian densities. From the results of the examples, it is

clear that the distribution function of T is not Gaussian under either hypothesis (see Figures 5–7). It is interesting to observe, though, that the probability of error P_e is very similar for the two tests, i.e., P_e is not very sensitive to the value of η. We note that the Gaussian approximation P_{Ge} of the probability of error is neither an upper nor a lower bound for P_e since sometimes it over predicts (by 21.9% in Example 1, $K = 10$) and sometimes it under predicts (by 11.3% in Example 2, $K = 1$) the value of P_e. Hence, P_{Ge} is not a reliable quantity for estimating P_e.

Further research should be devoted in studying the behavior of P_e as a function of the weights $\{w_k\}$, as a function of the sampling points $\{x_k\}$, and as a function of the total number of weights of the filter g. It is very important that fast and accurate methods be found in order to compute the functions $f_i(t)$ and $G_i(t)$, since straightforward applications of numerical integration are rather time consuming.

Appendix A
Evaluation of the Likelihood Ratio

In order to perform the test (2.6), we need to compute the "density" function of T under each hypothesis. Loosely speaking, this can be accomplished by computing the inverse Fourier transform of the characteristic function of T. Several methods have been proposed (see [3] and the references therein) for carrying out this computation. Our numerical solution relies on the use of the quadrature subroutines D01FCF and D01AMF from the NAG library.

For the purpose of computing the LR function and the probability of error for the test (2.6), we introduce the moment generating function of T, denoted by $M_i(s)$; it is given by [5, p. 381]

$$\begin{aligned} M_i(s) &\triangleq \mathsf{E}_i[e^{sT}] \\ &= \exp\left\{\int \lambda_i(x)[e^{s\tilde{g}(-x)} - 1]dx\right\}. \end{aligned} \tag{A.1}$$

From (2.5) it follows that

$$M_i(s) = e^{-\Lambda_i} + K_i(s), \tag{A.2}$$

where

$$K_i(s) \triangleq \int_{-\infty}^{\infty} e^{st} f_i(t)\, dt. \tag{A.3}$$

With $s = \sigma + j\omega$, let

$$\begin{aligned} C_i^{\sigma}(\omega) &\triangleq \mathrm{Re}\left\{\int \lambda_i(x)e^{s\tilde{g}(-x)}dx\right\} \\ &= \int \lambda_i(x)e^{\sigma\tilde{g}(-x)}\cos(\omega\tilde{g}(-x))\, dx, \end{aligned}$$

and

$$\begin{aligned} S_i^{\sigma}(\omega) &\triangleq \mathrm{Im}\left\{\int \lambda_i(x)e^{s\tilde{g}(-x)}dx\right\} \\ &= \int \lambda_i(x)e^{\sigma\tilde{g}(-x)}\sin(\omega\tilde{g}(-x))\, dx. \end{aligned}$$

We compute $C_i^{\sigma}(\omega)$ and $S_i^{\sigma}(\omega)$ numerically using the NAG routine D01FCF. If we set $s = j\omega$ in (A.3) and take inverse Fourier transforms, we obtain, since $\mathrm{Re}\{K_i(j\omega)e^{j\omega t}\}$ is an even function of ω,

$$f_i(t) = \frac{1}{\pi}\int_0^{\infty} \{e^{-(\Lambda_i - C_i(\omega))}\cos(S_i(\omega) - \omega t) - e^{-\Lambda_i}\cos(\omega t)\}\, d\omega, \tag{A.4}$$

where, for convenience of notation, we write C_i and S_i instead of C_i^{σ} and S_i^{σ} when $\sigma = 0$.

Appendix B
Evaluation of the Probability of Error

In order to compute P_e, we first need to compute (2.9). To this end, let $\hat{G}_i(s)$ denote the Laplace transform of G_i. More precisely,

$$\hat{G}_i(s) \triangleq \int_{-\infty}^{\infty} G_i(t)e^{st}dt, \qquad \mathrm{Re}\{s\} < 0. \tag{B.1}$$

Assuming that

$$\lim_{t\to-\infty} e^{st}G_i(t) = 0, \qquad \mathrm{Re}\{s\} < 0,$$

which is true in our applications, integration by parts yields

$$\hat{G}_i(s) = -\frac{1}{s}K_i(s). \tag{B.2}$$

Combining (B.2) with (A.2) and (A.1), we can substitute in (B.1) and take inverse Fourier transforms (after writing $s = \sigma + j\omega$, with $\sigma < 0$), to obtain

$$G_i(t) = -\frac{e^{-\sigma t}}{\pi}\int_0^{\infty} \frac{Q_i^{\sigma}(t,\omega)}{\sigma^2+\omega^2}\,d\omega, \tag{B.3}$$

where

$$\begin{aligned} Q_i^{\sigma}(t,\omega) = {} & e^{-(\Lambda_i - C_i^{\sigma}(\omega))}[\sigma\cos(S_i^{\sigma}(\omega)-\omega t) + \omega\sin(S_i^{\sigma}(\omega)-\omega t)] \\ & - e^{-\Lambda_i}(\sigma\cos(\omega t) - \omega\sin(\omega t)). \end{aligned}$$

The integrals (A.4) and (B.3) are computed numerically using the NAG routine D01AMF.

References

[1] W. A. Gardner, "A unifying view of second-order measures of quality for signal classification," *IEEE Trans. Commun.*, vol. COM-28, no. 6, pp. 807–816, June 1980.

[2] E. A. Geraniotis and H. V. Poor, "Minimax discrimination for observed Poisson processes with uncertain rate functions," *IEEE Trans. Inform. Theory*, vol. IT-31, no. 5, pp. 660–669, Sept. 1985.

[3] J. A. Gubner, "On the computation of shot-noise probability distributions," *IEEE Trans. Inform. Theory.*, submitted.

[4] G. M. Morris, "Scene matching using photon-limited images," *J. Opt. Soc. Am. A*, vol. 1, no. 5, pp. 482–488, May 1984.

[5] A. Papoulis, *Probability, Random Variables, and Stochastic Processes.* New York: McGraw-Hill, 1984.

[6] ——, "High density shot noise and Gaussianity," *J. Appl. Prob.*, vol. 8, pp. 118–127, 1971.

[7] H. V. Poor, *An Introduction to Signal Detection and Estimation.* New York: Springer-Verlag, 1988.

[8] R. E. Sequeira, *Binary Hypothesis Testing for Filtered Point Processes.* M. S. Thesis, University of Wisconsin–Madison. May 1990.

[9] D. L. Snyder, *Random Point Processes.* New York: Wiley, 1975.

[10] M. N. Wernick and G. M. Morris, "Image classification at low light levels," *J. Opt. Soc. Am. A*, vol. 3, no. 12, pp. 2179–2187, Dec. 1986.

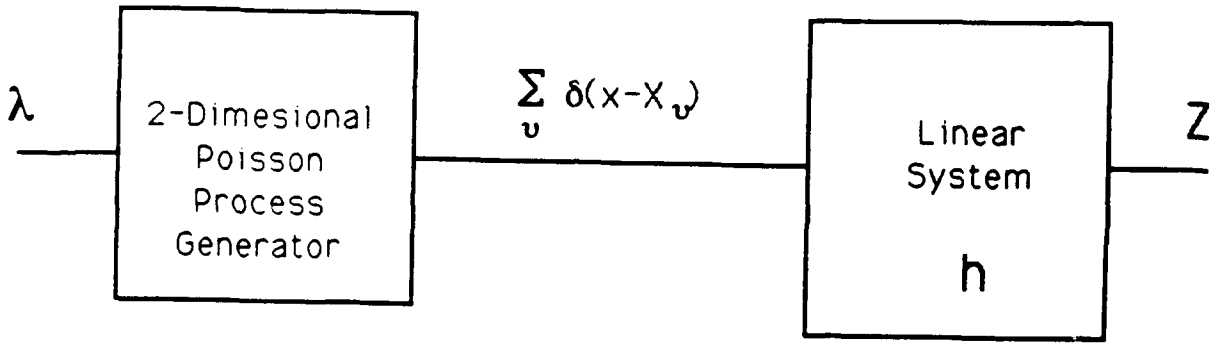


Fig. 1. Imaging system model.

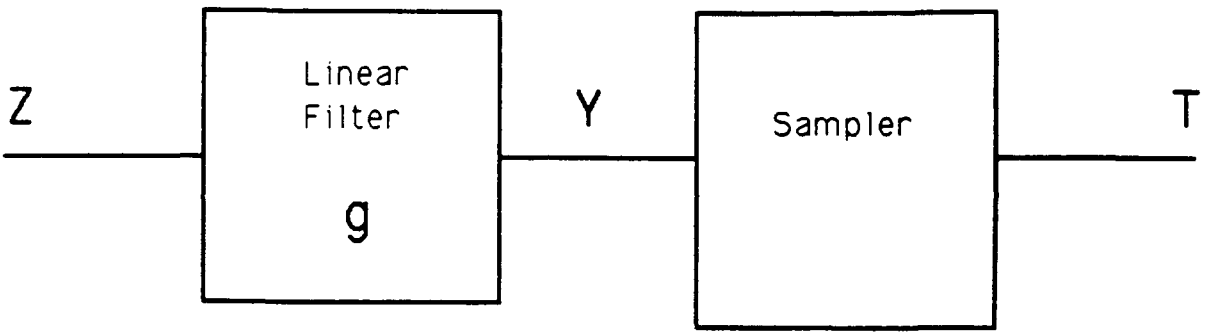
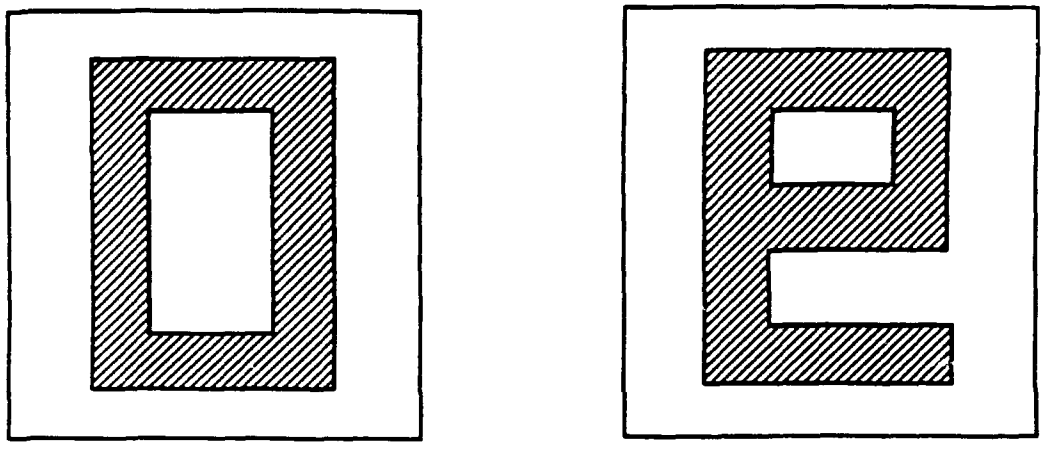


Fig. 2. Detector structure.



(a) (b)

Fig. 3. (a) Region for hypothesis 0. (b) Region for hypothesis 1.

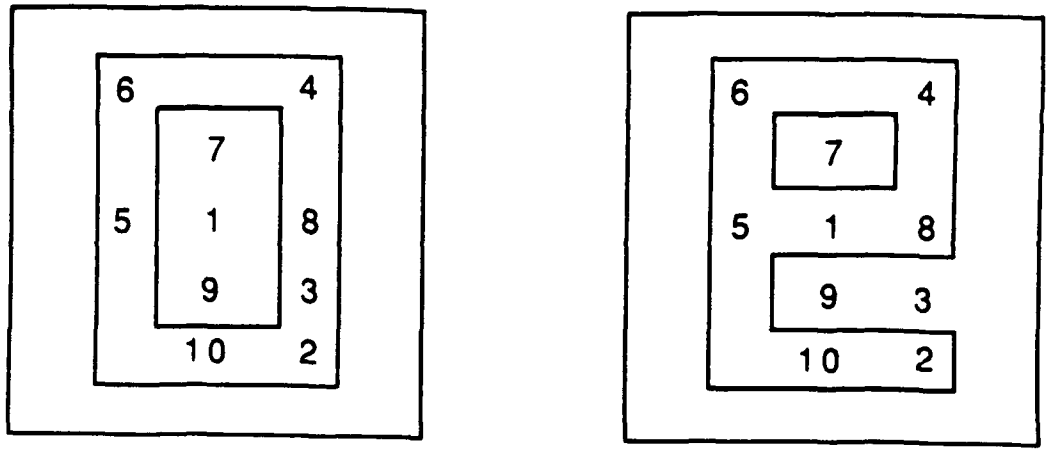
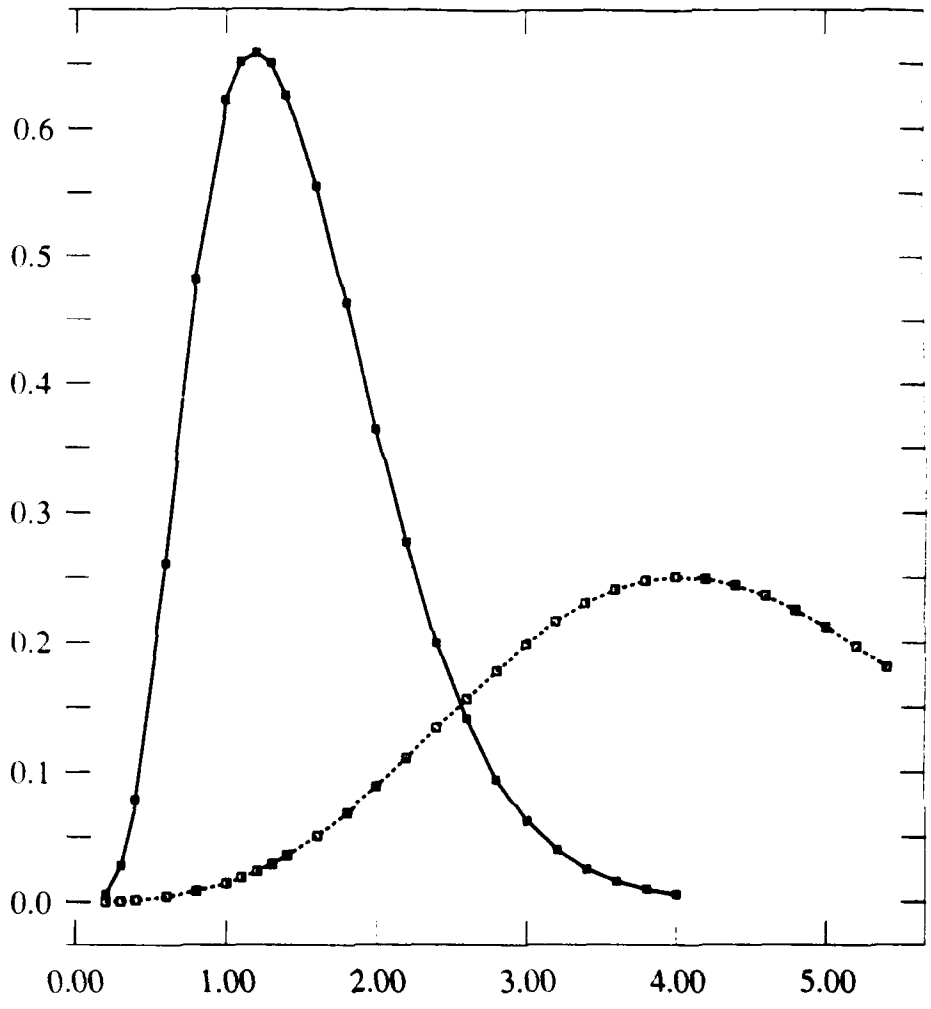
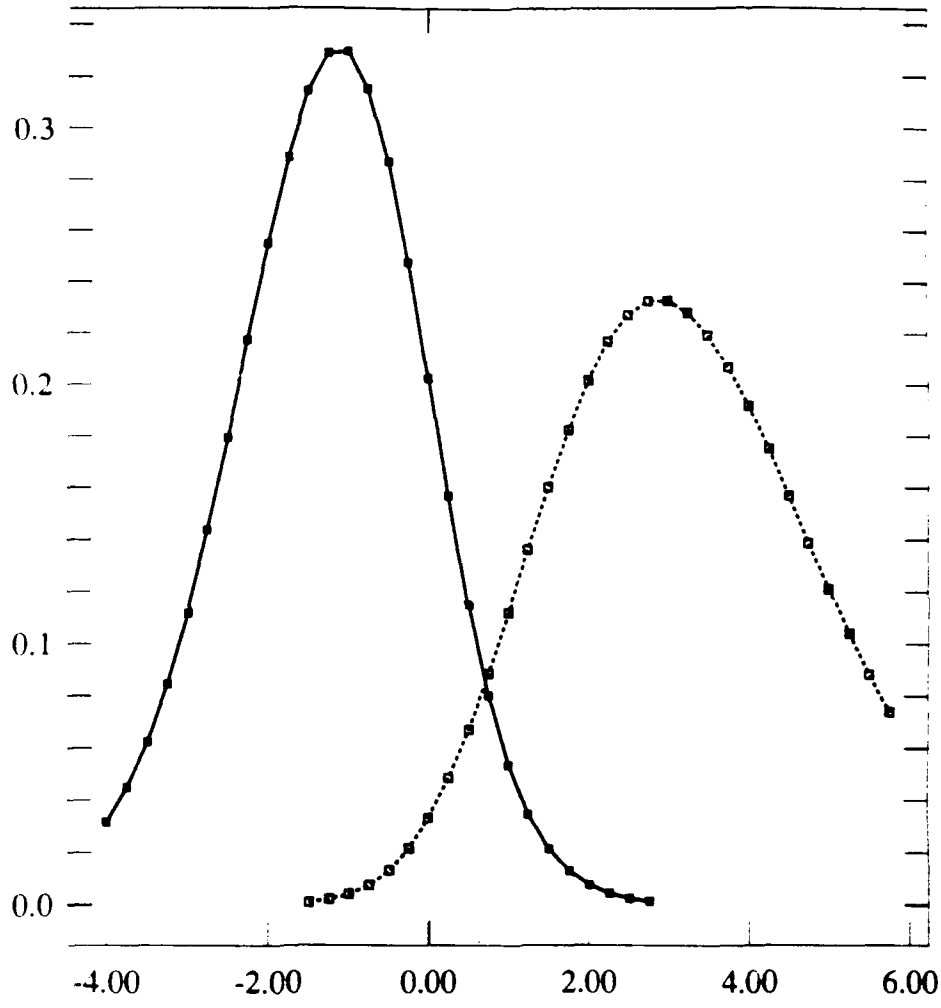


Fig. 4. The sampling points.



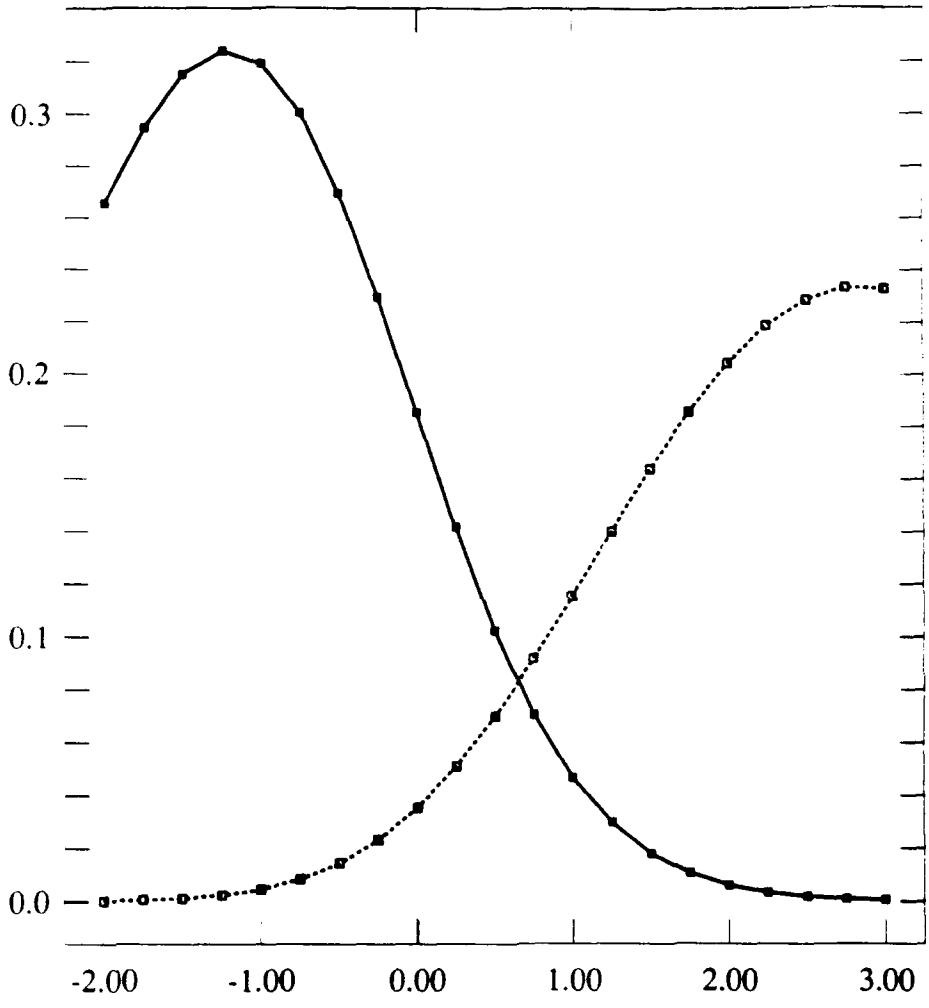
(a)

Fig. 5. Probability "density" function of T under H_0 (solid squares), and under H_1 (open squares) for Example 1. (a) $K = 1$.



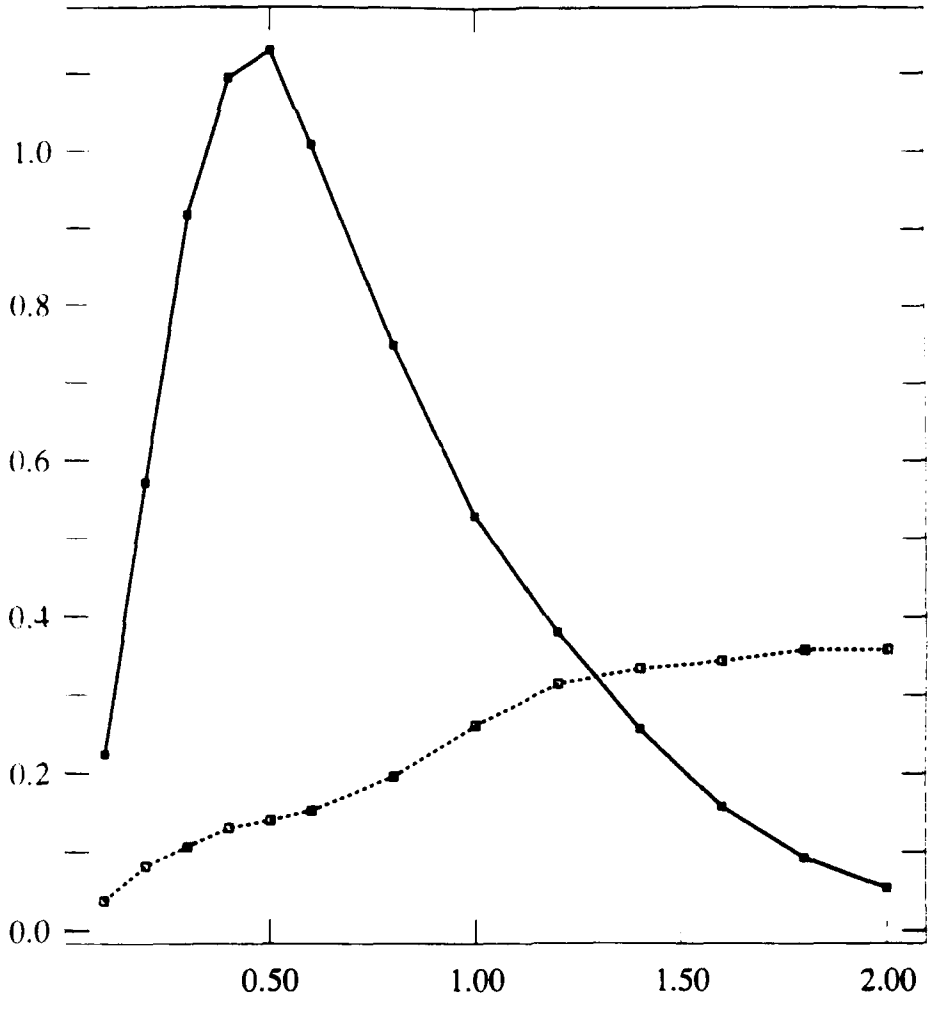
(b)

Fig. 5. cont. (b) $K = 5$.



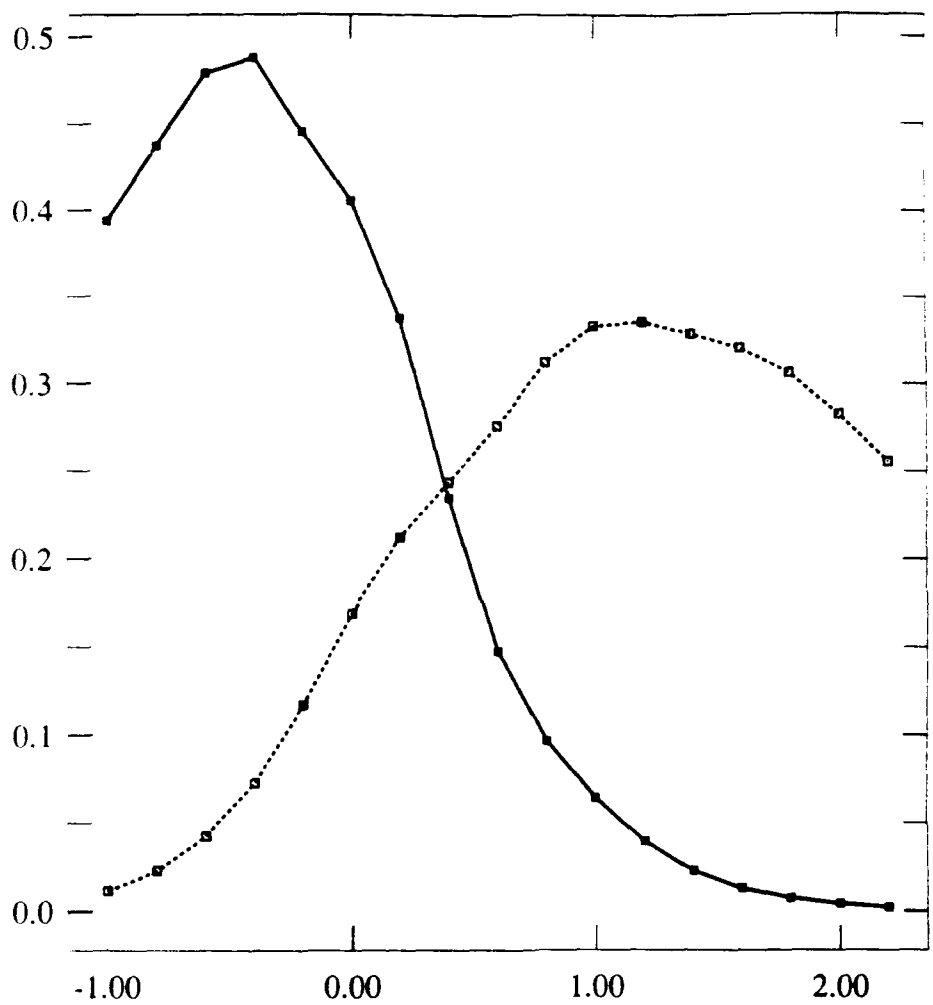
(c)

Fig. 5. cont. (c) $K = 10$.



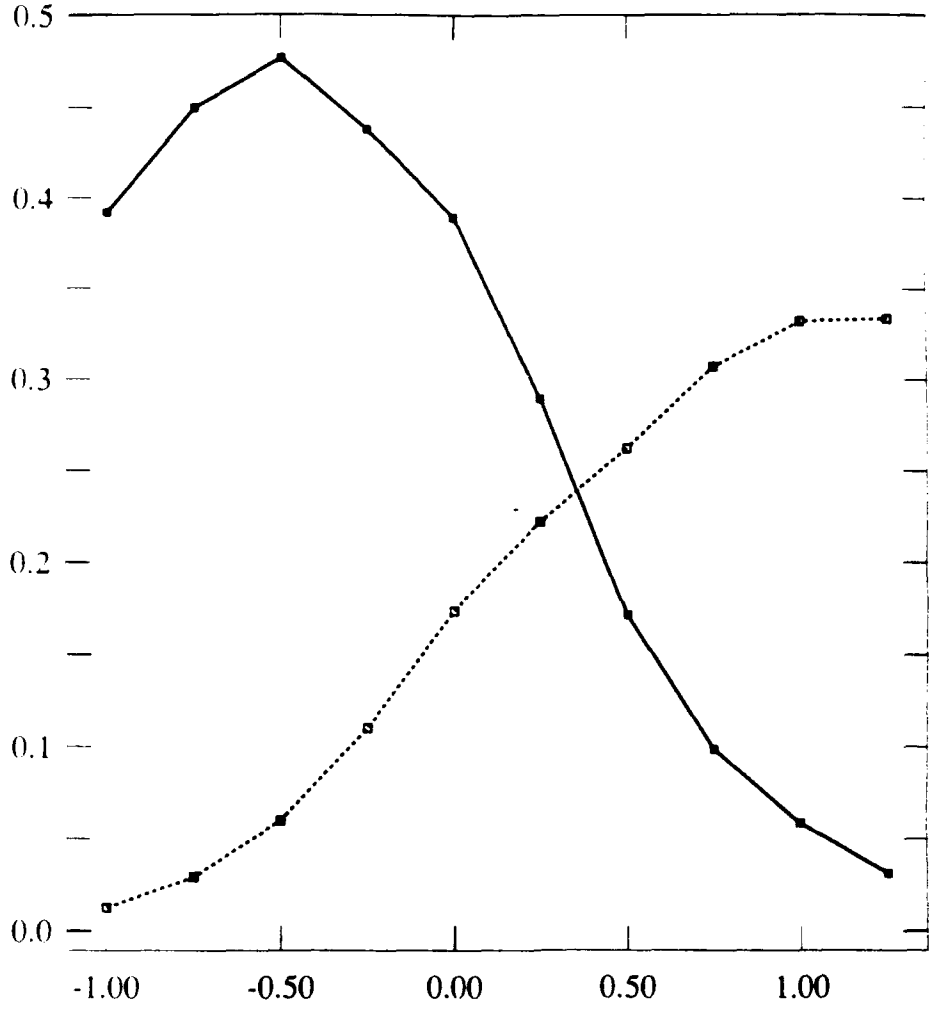
(a)

Fig. 6. Probability "density" function of T under H_0 (solid squares), and under H_1 (open squares) for Example 2. (a) $K = 1$.



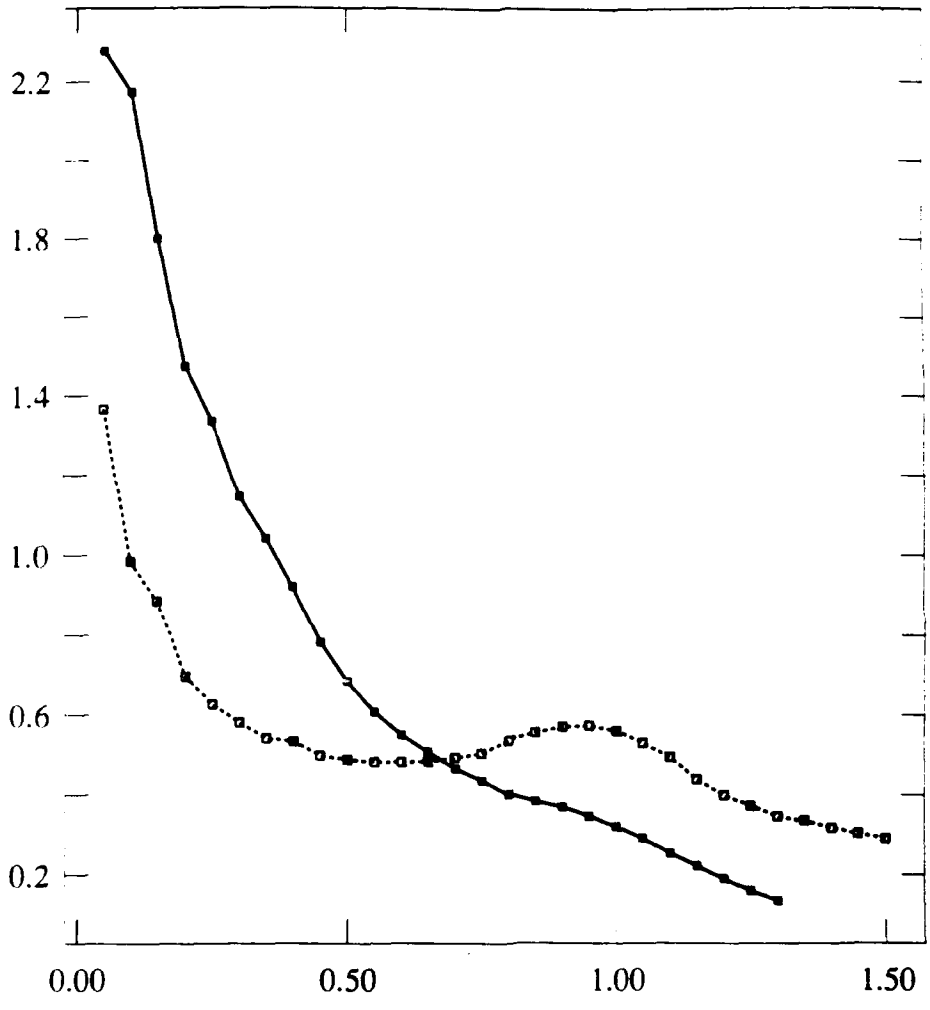
(b)

Fig. 6. cont. (b) $K = 5$.



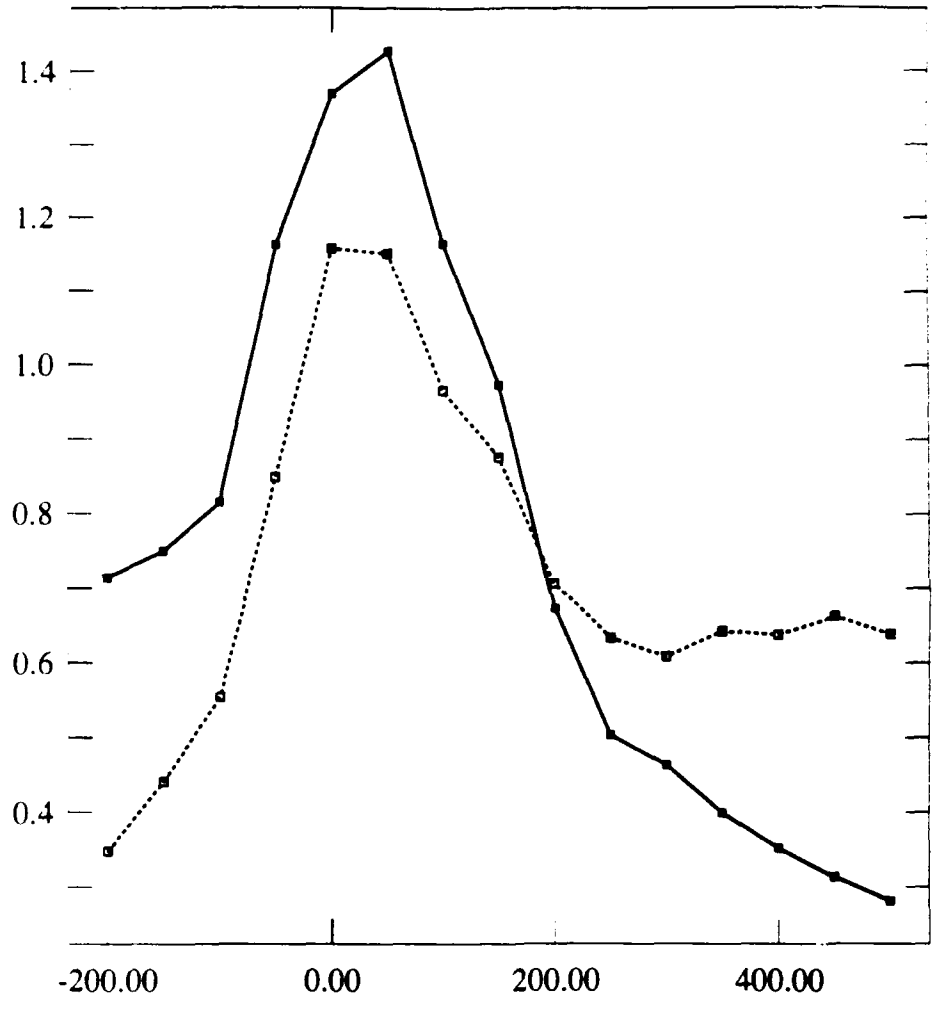
(c)

Fig. 6. cont. (c) $K = 10$.



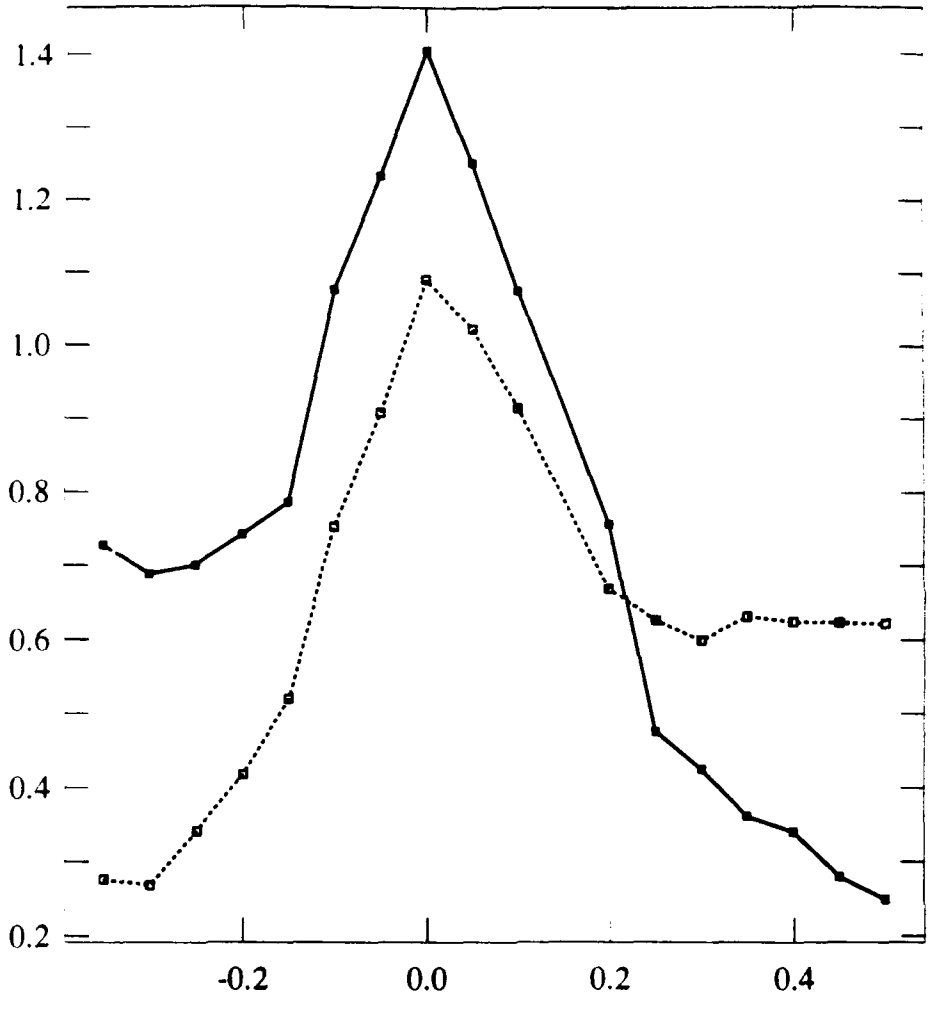
(a)

Fig. 7. Probability "density" function of T under H_0 (solid squares), and under H_1 (open squares) for Example 3. (a) $K = 1$.



(b)

Fig. 7. cont. (b) $K = 5$.



(c)

Fig. 7. cont. (c) $K = 10$.

Appendix E

Reprint of Reference [6]

Wavelet Transforms for Discrete-Time Periodic Signals

John A. Gubner, *Member, IEEE*, and Wei-Bin Chang

Abstract — A tutorial development of wavelet transforms for discrete-time periodic signals is presented using only basic concepts from linear algebra. Like the discrete Fourier transform, these wavelet transforms are shown to be unitary operators that exhibit frequency-localization properties. They are also shown to exhibit time-localization properties and to be well suited for signal-compression applications. Wavelet transforms for images, i.e., matrices, are briefly discussed. Wavelet transforms are constructed that are fast in the sense that their complexity is no greater than that of the corresponding fast Fourier transform.

I. INTRODUCTION

The study of wavelets has drawn considerable attention from the signal processing community since the appearance of the papers by Daubechies [2, 3] and Mallat [8, 9, 10]. Unfortunately, a complete understanding of their work requires a solid background in functional analysis. However, just as the discrete Fourier transform for finite-length data records can be rigorously derived using only concepts from linear algebra, we do the same for wavelet transforms in this tutorial. It is hoped that by restricting attention to this finite-dimensional framework, the presentation will be more accessible.

There are certain differences between continuous-time, infinite-dimensional wavelets and the finite-dimensional, discrete-time wavelets considered here. For example, the analog of $\psi_{j0}(t) = \sqrt{2}\,\psi_{j+1,0}(2t)$ [2, p. 958] does not hold here. Another difference is that in the infinite-dimensional case, one deals with a single pair of operators, G and H, while for discrete-time periodic signals we have a sequence of operators, G_j and H_j. Furthermore, instead of expressions such as $(G^*)^n z$ [2, p. 947], we consider expressions of the form $G_0^* \cdots G_{n-1}^* z$; in the former case, there is the recursion, $(G^*)^n z = G^*[(G^*)^{n-1} z]$, while for us there is no simple recursion to obtain $G_0^* \cdots G_{n-1}^* z$ from $G_0^* \cdots G_{n-2}^* z$. Fortunately, these differences do more to reveal the structure of wavelet transforms than to complicate our development.

Tutorials that discuss continuous-time wavelets and related topics can be found in [5, 6, 13, 15]. Extensive bibliographies are contained in [5, 6, 13].

This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-90-0181.

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.

A. What are Wavelets?

Let X_0 denote the space of discrete-time signals $x_0(n)$ with period N, where N is a power of 2. Before defining wavelets, recall that from the theory of discrete Fourier transforms, every signal $x_0 \in X_0$ can be expressed as

$$x_0(n) = \sum_{k=0}^{N-1} \langle x_0, E_k \rangle E_k(n),$$

where

$$E_k(n) \triangleq \frac{1}{\sqrt{N}} e^{j\frac{2\pi}{N}kn},$$

$$\langle x_0, y_0 \rangle \triangleq \sum_{n=0}^{N-1} x_0(n)\overline{y_0(n)},$$

and the overbar denotes the complex conjugate. The reason the decomposition works is that the signals $\{E_k\}_{k=0}^{N-1}$ form an orthonormal basis for X_0. While direct computation of the Fourier coefficients,

$$\langle x_0, E_k \rangle = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_0(n) e^{-j\frac{2\pi}{N}kn},$$

for $k = 0, \ldots, N-1$ would require N^2 multiplications, the fast Fourier transform can compute these coefficients with order $N \log_2 N$ multiplications.

In this paper we consider orthonormal "wavelet" bases for X_0 consisting of signals $\psi_{ji}(n)$ and $\varphi_{Ji}(n)$. For ψ_{ji}, $j = 1, \ldots, J$ and for each j, $i = 0, \ldots, N/2^j - 1$. For φ_{Ji}, $i = 0, \ldots, N/2^J - 1$. Since these signals form an orthonormal basis, every $x_0 \in X_0$ can be written in the form

$$x_0(n) = \sum_{j=1}^{J} \Bigg(\sum_{i=0}^{N/2^j-1} \langle x_0, \psi_{ji} \rangle \psi_{ji}(n) \Bigg) + \sum_{i=0}^{N/2^J-1} \langle x_0, \varphi_{Ji} \rangle \varphi_{Ji}(n). \tag{1}$$

The signals ψ_{ji} are called wavelets, and the signals φ_{Ji} are called scaling functions; examples for the case $N = 128$ and $J = 5$ are shown in Figs. 1–6. The numbers $\langle x_0, \psi_{ji} \rangle$ and $\langle x_0, \varphi_{Ji} \rangle$ are called wavelet coefficients, and the fast wavelet transform is an algorithm for computing them with fewer than N^2 multiplications; in some cases this can be done with order $N \log_2 N$ multiplications.

We now compare and contrast the sinusoidal basis functions E_k with the wavelets ψ_{ji} and φ_{Ji}. The constant function, $E_0(n) \equiv 1/\sqrt{N}$, exhibits no variation at all, while the

signals $\varphi_{J,i}(n)$ exhibit the least variation of all the wavelet signals. At the other extreme, the signal $E_{N-1}(n)$ exhibits the most variation of all the sinusoids $E_k(n)$, while the signals $\psi_{1,i}(n)$ exhibit the most variation of all the wavelet signals. One important difference between the wavelets in Figs. 1-6 and the sinusoids E_k is that the $\psi_{j,i}(n)$ are zero for most values of n. For example, in Fig. 1, we see that $\psi_{1,2}(n)$ is nonzero only for $n = 2, 3, 4, 5$. Thus, looking at wavelet coefficients can help us localize high-variation features in time rather precisely. Slower-varying signals cannot be localized as well.

B. Outline of the Paper

The remainder of the paper is organized as follows. In Section II we focus on the basic properties of pyramid-type algorithms [1], which will be the key to defining and computing wavelet transforms in Section III. In particular, we show that such algorithms are natural vehicles for frequency localization and for signal compression. We also show how these algorithms interact with shift operators to exhibit time-localization behavior.

In Section III we define wavelet transforms for discrete-time periodic signals. We then define the corresponding wavelet bases in Section III-A. In Section III-B we discuss fast wavelet transforms whose complexity is no greater than that of the corresponding fast Fourier transform. Numerical examples are presented in Section III-C, where we discuss frequency localization, time-frequency filtering, and signal compression using discrete wavelet transforms. In Section III-D we briefly sketch an extension of wavelet transforms to two-dimensional signals such as images.

In Section IV we extract the key ideas from [2, Sections 3.A. and 4.B.] to give a self-contained derivation of the sequence of operators used to implement fast wavelet transforms for discrete-time periodic signals.

II. Pyramid Algorithms

The basic building block for pyramid-type algorithms is a mapping from an input space X into a product space, say $U \times V$, as discussed in Section II-A. In Section II-B we iterate a sequence of these operators to obtain a Decomposition Algorithm and a Reconstruction Algorithm. Applications to signal compression and time localization are discussed in Sections II-C and II-D, respectively.

A. Isometries and Product Spaces

Let U and V be inner-product spaces with respective inner products $\langle \cdot, \cdot \rangle$ and $(\cdot, \cdot)$. This induces an inner product on $U \times V$ by taking

$$\left(\begin{bmatrix} u \\ v \end{bmatrix}, \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} \right) \triangleq \langle u, \tilde{u} \rangle + (v, \tilde{v}).$$

Now, if X is a third inner-product space, and if $G: X \to U$ and $H: X \to V$, are linear operators, we define the mapping $T: X \to U \times V$ by

$$Tx = \begin{bmatrix} G \\ H \end{bmatrix} x = \begin{bmatrix} Gx \\ Hx \end{bmatrix}. \tag{2}$$

Next, if the adjoint operators $G^*: U \to X$ and $H^*: V \to X$ exist, then the adjoint $T^*: U \times V \to X$ also exists and is given by

$$T^* \begin{bmatrix} u \\ v \end{bmatrix} = [G^* \; H^*] \begin{bmatrix} u \\ v \end{bmatrix} = G^*u + H^*v.$$

Our interest is in isometries; i.e., operators that preserve inner products, or equivalently, operators for which $T^*T = I$. For such operators, x can obviously be recovered from Tx.

Remark: If $T^*T = I$ and T is onto, then $TT^* = I$ as well [4, p. 185]. Such an operator is said to be unitary. Of course, if $\dim X = \dim U \times V < \infty$, then $T^*T = I$ always implies $TT^* = I$, since in this case T is 1-to-1 if and only if T is onto [7, p. 81].

For future reference note that since T has the form in (2), $T^*T = I$ is equivalent to

$$G^*G + H^*H = I, \tag{3}$$

and $TT^* = I$ is equivalent to the following three equations,

$$GG^* = I \tag{4}$$

$$HH^* = I \tag{5}$$

$$GH^* = 0. \tag{6}$$

To summarize, if we "decompose" x with

$$\begin{aligned} u &= Gx, \\ v &= Hx, \end{aligned} \tag{7}$$

and, if (3) holds, then we can reconstruct x from u and v with

$$x = G^*u + H^*v.$$

Remark: As shown in Section IV-C, in the wavelet setting, G can be viewed as a low-pass filter, and H can be viewed as a high-pass filter. Thus, in (7), we think of v as carrying the high-frequency detail in x, and we think of u as carrying the smoother, low-frequency information in x.

B. The General Algorithms

Let $X_0, X_1, \ldots, X_J$ and $V_1, \ldots, V_J$ be two sequences of inner-product spaces. Denote the inner product on X_j by $\langle \cdot, \cdot \rangle_j$ and the inner product on V_j by $(\cdot, \cdot)_j$. For $j = 0, \ldots, J-1$, let $G_j: X_j \to X_{j+1}$ and let $H_j: X_j \to V_{j+1}$. Consider the following procedure.

Decomposition Algorithm

Let $x_0 \in X_0$ be given.
for $j = 0$ **to** $J-1$
  $x_{j+1} = G_j x_j$
  $v_{j+1} = H_j x_j$
next j
end

If (3) holds for each pair G_j and H_j, then x_0 can be recovered from $v_1, \ldots, v_J, x_J$ by the following procedure.

Reconstruction Algorithm

Let $v_1, \ldots, v_J, x_J$ be given.
for $j = J-1$ **to** 0
  $x_j = G_j^* x_{j+1} + H_j^* v_{j+1}$
next j
end

If we let

$$Y \triangleq V_1 \times \cdots \times V_J \times X_J,$$

and if $v_1, \ldots, v_J$, and x_J are generated by the Decomposition Algorithm, then setting

$$Dx_0 = (v_1, \ldots, v_J, x_J) \tag{8}$$

defines a mapping $D: X_0 \to Y$. We equip Y with the induced inner product, denoted by $(\cdot,\cdot)$. Note that we are now using row-vector notation for product-space vectors.

Remark: We can extend the ideas in the remark at the end of Section II-A to D by saying that v_1 contains the highest frequency information in x_0, while v_J and x_J contain the lowest frequency information about x_0.

Proposition 1: Assuming only that G_j^* and H_j^* exist, the vector $\tilde{x}_0$ generated by applying the Reconstruction Algorithm to an arbitrary vector $\tilde{y} = (\tilde{v}_1, \ldots, \tilde{v}_J, \tilde{x}_J) \in Y$, is equal to $D^*\tilde{y}$; i.e., the Reconstruction Algorithm implements D^*.

Proof: Let x_0 be any element of X_0, and let $\tilde{y}$ be any element of Y. We must show that if $\tilde{x}_0$ is obtained by applying the Reconstruction Algorithm to $\tilde{y}$, then $(Dx_0, \tilde{y}) = \langle x_0, \tilde{x}_0 \rangle_0$. Write

$$(Dx_0, \tilde{y}) = \sum_{j=1}^{J} (v_j, \tilde{v}_j)_j + \langle x_J, \tilde{x}_J \rangle_J, \tag{9}$$

where $v_1, \ldots, v_J$, and x_J are obtained by applying the Decomposition Algorithm to x_0. In addition to $\tilde{x}_0$, let $\tilde{x}_1, \ldots, \tilde{x}_{J-1}$ also be obtained by applying the Reconstruction Algorithm to $\tilde{y}$. Then for each $j = J, \ldots, 1$, substitute the following identity into (9):

$$\begin{aligned}
&(v_j, \tilde{v}_j)_j + \langle x_j, \tilde{x}_j \rangle_j \\
&\quad = (H_{j-1}x_{j-1}, \tilde{v}_j)_j + \langle G_{j-1}x_{j-1}, \tilde{x}_j \rangle_j \\
&\quad = \langle x_{j-1}, H_{j-1}^* \tilde{v}_j \rangle_{j-1} + \langle x_{j-1}, G_{j-1}^* \tilde{x}_j \rangle_{j-1} \\
&\quad = \langle x_{j-1}, H_{j-1}^* \tilde{v}_j + G_{j-1}^* \tilde{x}_j \rangle_{j-1} \\
&\quad = \langle x_{j-1}, \tilde{x}_{j-1} \rangle_{j-1}.
\end{aligned}$$

□

Corollary 2: If (3) holds for each pair G_j and H_j, then $D^*D = I$; i.e., D is an isometry.

Proof: If we had assumed that $\tilde{y} = D\tilde{x}_0$ in the preceding proof, then we would have obtained $(Dx_0, D\tilde{x}_0) = \langle x_0, \tilde{x}_0 \rangle_0$; i.e., D preserves inner products. □

Corollary 3: If (4)-(6) hold for each pair G_j and H_j, then $DD^* = I$. If (3) also holds, then D is unitary.

Proof: Fix a $y = (v_1, \ldots, v_J, x_J) \in Y$ and let $x_{J-1}, \ldots, x_0$ be generated by the Reconstruction Algorithm. Now apply the Decomposition Algorithm to x_0. Since $x_0 = G_0^* x_1 + H_0^* v_1$, $G_0 x_0 = x_1$ by (4) and (6). Similarly, for $j = 1, \ldots, J-1$, $G_j x_j = x_{j+1}$. In addition, since $G_j H_j^* = 0$ implies $H_j G_j^* = 0$,

$$H_j x_j = H_j(G_j^* x_{j+1} + H_j^* v_{j+1}) = v_{j+1}.$$

Hence, $DD^* y = y$. □

Proposition 4: If D is unitary, then we can write X_0 as

$$X_0 = W_1 \oplus \cdots \oplus W_J \oplus B_J,$$

where

$$W_j \triangleq \{D^* y : y = (0, \ldots, 0, v_j, 0, \ldots, 0), v_j \in V_j\}$$

and

$$B_J \triangleq \{D^* y : y = (0, \ldots, 0, x_J), x_J \in X_J\}$$

are orthogonal subspaces of X_0.

Proof: Since $D^*D = I$, we can write $x_0 = D^*(Dx_0)$, and by (8) it is clear that X_0 is a sum of the indicated subspaces. Since $DD^* = I$, it is easy to see that X_0 is a direct sum and that the subspaces are orthogonal. □

C. Application to Signal Compression

Given $x_0 \in X_0$, apply the Decomposition Algorithm to obtain $Dx_0 \in Y$ as in (8). Assuming that (3) holds for each G_j and H_j, D preserves norms, and thus

$$\begin{aligned}
\|x_0\|_0^2 &= \|Dx_0\|^2 \\
&= \sum_{j=1}^{J} \|v_j\|_j^2 + \|x_J\|_J^2.
\end{aligned}$$

To store a compressed version of x_0, we save only those v_j and x_J of significant energy relative to $\|x_0\|_0^2$. To recover an estimate of x_0, we apply the Reconstruction Algorithm, replacing the omitted data with zeros. Numerical examples are discussed in Section III-C.

D. Interaction with "Shift" Operators—Time Localization

Fix an integer k, $1 \le k \le J$, and consider a sequence of operators $S_0, \ldots, S_k$ such that $S_j: X_j \to X_j$ and such that

$$G_j S_j = S_{j+1} G_j, \qquad j = 0, \ldots, k-1. \tag{10}$$

Remark: When this setup is applied to wavelets, we take X_j to be the set of column vectors of length $N/2^j$, where N is a power of 2. We interpret S_0 as a circular shift of 2^k units, and S_j as a shift of 2^{k-j} units. Thus S_k would be a shift of $2^0 = 1$ unit. See Section III for further discussion.

Fix $x_0 \in X_0$, and let $x_1, v_1, \ldots, x_J, v_J$ be given by the Decomposition Algorithm. Set $\tilde{x}_0 \triangleq S_0 x_0$ and let $\tilde{x}_1, \tilde{v}_1, \ldots, \tilde{x}_J, \tilde{v}_J$ be generated by applying the Decomposition Algorithm to $\tilde{x}_0$. Then it follows by induction that

$$\tilde{x}_j = S_j x_j, \quad j = 0, \ldots, k. \tag{11}$$

At this point we restrict our setup and require that the space $V_{j+1} = X_{j+1}$, $j = 0, \ldots, J-1$, so that S_{j+1} can operate on elements of $V_{j+1} = X_{j+1}$. We assume that in addition to (10),

$$H_j S_j = S_{j+1} H_j, \qquad j = 0, \ldots, k-1. \tag{12}$$

With x_j, v_j and $\tilde{x}_j, \tilde{v}_j$ as before, we have, for $j = 0, \ldots, k-1$,

$$\begin{aligned} \tilde{v}_{j+1} &\triangleq H_j \tilde{x}_j \\ &= H_j S_j x_j, \qquad \text{by (11)}, \\ &= S_{j+1} H_j x_j, \qquad \text{by (12)}, \\ &= S_{j+1} v_{j+1}. \end{aligned} \tag{13}$$

Now consider an x_0 such that applying the Decomposition Algorithm to x_0 yields

$$Dx_0 = (0, \ldots, 0, v_k, 0, \ldots, 0).$$

We claim that if (3) holds for each pair G_j and H_j, then

$$D\tilde{x}_0 = D(S_0 x_0) = (0, \ldots, 0, S_k v_k, 0, \ldots, 0).$$

By (13) $\tilde{v}_1, \ldots, \tilde{v}_{k-1}$ are all zero and $\tilde{v}_k = S_k v_k$. If $k = J$, simply note that $\tilde{x}_J = 0$ by (11). If $k < J$, the Reconstruction Algorithm shows (on account of (3)) that $x_J, \ldots, x_k$ are all zero. Then by (11), $\tilde{x}_k = 0$. Now, by the Decomposition Algorithm, $\tilde{x}_{k+1}, \tilde{v}_{k+1}, \ldots \tilde{x}_J, \tilde{v}_J$ are all zero.

In general we have

$$D(S_0^l x_0) = (0, \ldots, 0, S_k^l v_k, 0, \ldots, 0).$$

Remark: If D is unitary, we see that S_0 maps W_k into itself. If $k = J$, S_0 also maps X_J into itself.

III. Discrete Wavelet Transforms

To make the foregoing discussion concrete, we apply it to the following situation. Let N be a power of 2, and set

$$X_j \triangleq \mathbb{C}^{N/2^j}, \qquad j = 0, \ldots, J,$$

where $2 \le N/2^J$ and $\mathbb{C}^{N/2^j}$ is equipped with the usual Euclidean inner product. We take $V_j = X_j$, $j = 1, \ldots, J$, as was done in Section II-D.

Convention: As in the theory of discrete Fourier transforms, we do not distinguish between vectors in $\mathbb{C}^{N/2^j}$ and their doubly-infinite $N/2^j$-periodic extensions. Recall that the N-length discrete Fourier transform (DFT) of a sequence $x \in \mathbb{C}^N$ is defined by

$$\hat{x}(u) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nu},$$

for all integers u.

Definition 5 (Discrete Wavelet Transform): If the operators $G_j: X_j \to X_{j+1}$ and $H_j: X_j \to V_{j+1}$ have the form

$$(G_j x)(m) = \sum_{n=0}^{N/2^j-1} g_j(n-2m)x(n), \tag{14}$$

$$(H_j x)(m) = \sum_{n=0}^{N/2^j-1} h_j(n-2m)x(n), \tag{15}$$

where g_j, h_j, and x are $N/2^j$-length periodic sequences, and where g_j and h_j are such that (3)–(6) hold for G_j and H_j, then Dx_0 defined by using these operators in the Decomposition Algorithm is called a discrete wavelet transform (DWT) of x_0.

For operators of the form (14) and (15), if $1 \le k \le J$, and if

$$(S_j x)(n) \triangleq x(n - 2^{k-j}), \quad j = 0, \ldots, k,$$

where $x \in X_j = \mathbb{C}^{N/2^j}$, then the periodicity convention can easily be used to show that (10) and (12) hold.

Remarks: (*i*) From the derivation in Section IV-A and from (57) in the Appendix, it can be seen that if G_j and H_j satisfy (3)–(6), then g_j and h_j are discrete-time periodic analogs of conjugate quadrature filters in [14]. In Section IV we construct g_j and h_j under two additional constraints. The first is that $g_j(n)$ and $h_j(n)$ be zero for most values of n. The second is that Daubechies' constraint (51) hold.

(*ii*) The Decomposition and Reconstruction Algorithms can be implemented by analysis and synthesis filter banks, respectively.

(*iii*) The operators G_j and H_j in (14) and (15) can be identified with $(N/2^{j+1}) \times (N/2^j)$ matrices; for an example with $j = 0$, see (28)–(29) below.

A. Wavelet Bases

We now construct a special orthonormal basis for X_0. Recalling Proposition 4, it suffices to construct orthonormal bases for W_j and B_J.

For each space $\mathbb{C}^{N/2^j}$, let δ_j denote the $N/2^j$-periodic Kronecker delta, and set $\delta_{ji}(n) \triangleq \delta_j(n-i)$. Then the δ_{ji}, $i = 0, \ldots, N/2^j - 1$, constitute the standard basis for $\mathbb{C}^{N/2^j} = X_j = V_j$. Next, set

$$d_{ji} \triangleq (0, \ldots, \delta_{ji}, \ldots, 0), \quad j = 1, \ldots, J,$$

where δ_{ji} is in the jth position out of $J+1$ positions. Set

$$e_{Ji} \triangleq (0, \ldots, 0, \delta_{Ji}).$$

If we let

$$\psi_{ji} \triangleq D^* d_{ji} \quad \text{and} \quad \varphi_{Ji} \triangleq D^* e_{Ji},$$

and if we assume that D is unitary, then

$$\{\psi_{j0}, \ldots, \psi_{j,N/2^j-1}\}$$

is an orthonormal basis for W_j, and

$$\{\varphi_{J0}, \ldots, \varphi_{J,N/2^J-1}\}$$

is an orthonormal basis for B_J. The union of all these bases is called a wavelet basis for X_0. The signals ψ_{ji} are called wavelets, and the signals φ_{Ji} are called scaling functions. Examples are discussed in Section III-C.

Remarks: (*i*) Recalling the expansion (1), observe that

$$\begin{aligned} \langle x_0, \psi_{ji} \rangle_0 &= \langle x_0, D^* d_{ji} \rangle_0 \\ &= (Dx_0, d_{ji}) \\ &= (v_j, \delta_{ji})_j \\ &= v_j(i). \end{aligned}$$

Similarly, $\langle x_0, \varphi_{Ji} \rangle_0 = x_J(i)$.

(*ii*) By the results of Section II-D, we see that for $k = 1, \ldots, J$,

$$\psi_{ki}(n - l2^k) = \psi_{k,i+l}(n), \quad l = 0, \ldots, N/2^k - 1, \quad (16)$$

and

$$\varphi_{Ji}(n - l2^J) = \varphi_{J,i+l}(n), \quad l = 0, \ldots, N/2^J - 1. \quad (17)$$

In particular, it follows that the subspace W_k (resp. B_J) is closed under cyclic shifts of 2^k (resp. 2^J) units.

(*iii*) The Reconstruction Algorithm shows that $\psi_{1i} = H_0^* \delta_{1i}$, $\psi_{2i} = G_0^* H_1^* \delta_{2i}$, and

$$\psi_{ji} = G_0^* \cdots G_{j-2}^* H_{j-1}^* \delta_{ji}, \quad j = 2, \ldots, J.$$

Also,

$$\varphi_{Ji} = G_0^* \cdots G_{J-1}^* \delta_{Ji}. \quad (18)$$

B. Fast Wavelet Transforms

Let p be an even integer such that $2 \leq p \leq N/2^J$. As will be shown in Section IV, one can find a sequence $g(0), \ldots, g(p-1)$ such that if

$$g_j(n) \triangleq \begin{cases} g(n), & n = 0, \ldots, p-1, \\ 0, & n = p, \ldots, N/2^j - 1, \end{cases} \quad (19)$$

and if

$$h_j(n) \triangleq (-1)^n \overline{g_j(1-n)}, \quad n = 0, \ldots, N/2^j - 1, \quad (20)$$

where $j = 0, \ldots, J$, then the operators G_j and H_j given by (14) and (15), respectively, satisfy (3)-(6). The discrete wavelet transform is then implemented by the Decomposition Algorithm, and the inverse transform is implemented by the Reconstruction Algorithm.

We now show that for $2p \leq \log_2 N$, such a discrete wavelet transform is no more difficult to compute than an N-point fast Fourier transform, which requires order $N \log_2 N$ complex multiplications. Since $g_j(n)$ and $h_j(n)$ have only p nonzero terms, the number of multiplications required to compute (14) and (15) for $m = 0, \ldots, N/2^{j+1} - 1$ is $pN/2^j$. To implement the Decomposition Algorithm requires that we do this for $j = 0, \ldots, J-1$. Hence, the total number of multiplications is always less than $2pN$. If $2p \leq \log_2 N$, then $2pN \leq N \log_2 N$. Note that even if $2p > \log_2 N$, we can still use the inequality $p \leq N/2^J$ to bound the number of multiplications by $N^2/2^{J-1}$, which is less than the N^2 multiplications required to compute directly all of the N-length inner products appearing in (1).

In fact, further speed-ups can be achieved by adapting some of the ideas in Rioul and Duhamel [12, Section III]. For example, at each interation j, the sums in (14) and (15) can be split into sums over n = even and n = odd. This results in the sum of two convolutions. Depending on the relative values of p and $\log_2(N/2^{j+1})$, it may be more efficient to do this with fast Fourier transforms. For small p, Rioul and Duhamel suggest the use of short-length fast running FIR algorithms [11, 16].

C. Numerical Examples

To obtain the $g(n)$ mentioned at the beginning of the previous subsection, it is necessary to solve (50)-(51) in Section IV. We take $p = 4$ so that these equations can easily be solved by hand. In the course of the calculations, there are two places where square roots must be taken. Thus, there are four possible solutions, depending on whether positive or negative roots are used. If the negative root is always taken, we obtain

$$\begin{aligned} g(0) &= .482962913145 \\ g(1) &= .836516303738 \\ g(2) &= .224143868042 \\ g(3) &= -.129409522551 \end{aligned} \quad (21)$$

These are the first four entries in [2, Table 1, p. 980] (where they are called $h(n)$). If we had always taken the positive square root, we would have obtained $\tilde{g}(n) = g(3-n)$, $n = 0, 1, 2, 3$. The two remaining solutions are $-g(n)$ and $-\tilde{g}(n)$.

Let g be given by (21) and define g_j and h_j by (19) and (20) with $N = 128$ and $J = 5$. Note that since g_j and h_j are real, if x_0 is real, so is Dx_0.

Example 1 (Wavelet Basis): The signals $\psi_{1,2}$, $\psi_{2,2}$, $\psi_{3,2}$, $\psi_{4,2}$, $\psi_{5,2}$, and $\varphi_{5,1}$ are shown in Figs. 1–6, respectively. The remaining basis signals can be obtained by translation as indicated in (16) and (17).

From the figures, one might suspect that

$$v_{j,2}(n) = \begin{cases} \sqrt{2}\,w_{j+1,2}(2n), & w_{j+1,2}(2n) \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

However, closer inspection reveals that this cannot be true. To see this, note that from Figs. 1 and 2, $v_{2,2}$ has 10 nonzero components, while $v_{1,2}$ has only 4 nonzero components. To more fully understand what is happening, it is necessary to consider the limiting continuous-time case as in [2].

Example 2 (Frequency Localization): As a low-frequency signal we consider

$$x_0(n) = \cos\left(\frac{2\pi}{128}7n\right), \quad n = 0, \ldots, 127, \tag{22}$$

and as a high-frequency signal we consider

$$x_0(n) = \cos\left(\frac{2\pi}{128}53n\right), \quad n = 0, \ldots, 127. \tag{23}$$

The signals are plotted in Figs. 7 and 8, and their DWTs are shown in Figs. 9 and 10.

Note: In the current setup, if $y = Dx_0$, then

$$y \in \mathbb{R}^{64} \times \mathbb{R}^{32} \times \mathbb{R}^{16} \times \mathbb{R}^{8} \times \mathbb{R}^{4} \times \mathbb{R}^{4}.$$

Hence, when examining graphs of $y(n)$, one should keep in mind that $y(0), \ldots, y(63)$ correspond to $v_1(0), \ldots, v_1(63)$, $y(64), \ldots, y(95)$ correspond to $v_2(0), \ldots, v_2(31)$, and so on.

Clearly, most of the energy in the low-frequency signal is located in v_3 and v_4, while most of the energy in the high-frequency signal is located in v_1.

Example 3 (Time-Frequency Filtering): Let x_0 denote the sum of the two signals in (22) and (23). We wish to remove the high-frequency signal during the time interval $n = 16, \ldots, 63$. Let $(v_1, \ldots, v_5, x_5)$ denote the DWT of x_0. Considering the sum of the DWTs in Figs. 9 and 10, most of the energy of the high-frequency signal is in v_1. Keeping in mind that v_1 is a subsampled signal of length $64 = 128/2$, and that $16/2 = 8$ and that $63/2$ truncates to 31, we set $v_1(8), \ldots, v_1(31)$ to zero. If we inverse DWT, we obtain the signal shown in Fig. 11.

To remove the low-frequency signal during a specified time interval would be more difficult because the time resolution of v_j for larger j is not as fine.

Example 4 (Signal Compression): We now apply the ideas of Section II-C to the DWT shown in Fig. 9. If we set $v_1 = 0$; i.e., we set the first 64 elements in the figure to zero for a 2-to-1 compression ratio, and if we apply the Reconstruction Algorithm, we obtain the waveform in Fig. 12.

In Fig. 10 we set $v_2 = 0$; i.e., we set elements 64-95 to zero for a 4-to-3 compression ratio. Applying the Reconstruction Algorithm, we obtain Fig. 13.

As a final example of signal compression, we consider the Gaussian pulse,

$$x_0(n) = \exp\left[-\left(\frac{n-63}{10}\right)^2\right], \quad n = 0, \ldots, 127, \tag{24}$$

shown in Fig. 14. Its DWT is shown in Fig. 15. If we set both v_1 and v_2 to zero, for a compression ratio of 4-to-1, and apply the Reconstruction Algorithm, we obtain the result shown in Fig. 16.

We close this subsection by noting that improved results can be obtained at the expense of using a larger value of p.

D. Discrete Wavelet Transforms for Images

We now briefly sketch a simple extension of one-dimensional wavelet transforms to handle two-dimensional images; i.e., where before x_0 was an $N \times 1$ vector, we now suppose that x_0 is an $N \times \tilde{N}$ matrix, where N and $\tilde{N}$ are both powers of 2. Let G and H be $N/2 \times N$ matrices that satisfy (3)–(6). Similarly, let $\tilde{G}$ and $\tilde{H}$ be $\tilde{N}/2 \times \tilde{N}$ matrices that also satisfy (3)–(6). Set

$$T \triangleq \begin{bmatrix} G \\ H \end{bmatrix} \quad \text{and} \quad \tilde{T} \triangleq \begin{bmatrix} \tilde{G} \\ \tilde{H} \end{bmatrix}.$$

By our assumptions on G, H and $\tilde{G}$, $\tilde{H}$, we have $T^*T = I$, $TT^* = I$, $\tilde{T}^*\tilde{T} = I$, and $\tilde{T}\tilde{T}^* = I$. Thus, the mapping $x_0 \rightarrow Tx_0\tilde{T}^*$ is a unitary operator.

Now observe that

$$Tx_0\tilde{T}^* = \begin{bmatrix} Gx_0\tilde{G}^* & Gx_0\tilde{H}^* \\ Hx_0\tilde{G}^* & Hx_0\tilde{H}^* \end{bmatrix}. \tag{25}$$

Let $x_1 = Gx_0\tilde{G}^*$ and let v_1 denote the remaining blocks of the matrix in (25). It should now be clear how to write

the Decomposition and Reconstruction Algorithms using a sequence of matrices G_j, H_j, $\tilde{G}_j$, and $\tilde{H}_j$ that satisfy (3)-(6). If these matrices are obtained as in Section III-B, then we require that $2 \le p, \tilde{p} \le \min\{N, \tilde{N}\}/2^J$.

IV. Construction of G_j and H_j for Fast Wavelet Transforms

As we saw in Section III-B, the key to fast wavelet transforms is to find g_j such that most of the $g_j(n)$ are zero. Then by (20) most of the $h_j(n)$ will also be zero. Of course, we also need to show that (3)-(6) hold for the corresponding operators G_j and H_j.

Our plan is to construct G_j from G_0 and then to construct H_j from G_j.

A. Obtaining H_j from G_j

Without loss of generality we may work with G_0 and H_0. To simplify the notation, we drop the subscript 0, and we let $M = N/2$. Then (14) and (15) become

$$(Gx)(m) = \sum_{n=0}^{N-1} g(n-2m)x(n), \tag{26}$$

$$(Hx)(m) = \sum_{n=0}^{N-1} h(n-2m)x(n), \tag{27}$$

where g, h, and x are N-periodic sequences. The purpose of this subsection is to exploit the formulas (26) and (27) to show that if (4) holds and if h is given by (46) below, then (3), (5), and (6) also hold. The derivation is adapted from [2, Section 3.A.].

Clearly, G and H can be identified with the $M \times N$ matrices

$$G = \begin{bmatrix} g(0) & g(1) & \cdots & g(N-1) \\ g(-2) & g(-1) & & g(N-3) \\ \vdots & & \ddots & \vdots \\ g(2-N) & g(3-N) & \cdots & g(1) \end{bmatrix} \tag{28}$$

$$H = \begin{bmatrix} h(0) & h(1) & \cdots & h(N-1) \\ h(-2) & h(-1) & & h(N-3) \\ \vdots & & \ddots & \vdots \\ h(2-N) & h(3-N) & \cdots & h(1) \end{bmatrix}. \tag{29}$$

Regarding (3)-(6) as matrix equations involving (28) and (29) we proceed as follows. First, consider the left-most columns of (4)-(6). We find that

$$\sum_{n=0}^{N-1} g(n-2m)\bar{g}(n) = \delta(m), \tag{30}$$

$$\sum_{n=0}^{N-1} h(n-2m)\bar{h}(n) = \delta(m), \tag{31}$$

$$\sum_{n=0}^{N-1} g(n-2m)\bar{h}(n) = 0, \tag{32}$$

where the overbar denotes the complex conjugate, and δ denotes the M-periodic Kronecker delta. If we define the M-periodic sequences $g_e(k) = g(2k), g_o(k) = g(2k+1), h_e(k) = h(2k)$, and $h_o(k) = h(2k+1)$, then (30)-(32) become

$$\sum_{k=0}^{M-1} g_e(k-m)\bar{g}_e(k) + g_o(k-m)\bar{g}_o(k) = \delta(m), \tag{33}$$

$$\sum_{k=0}^{M-1} h_e(k-m)\bar{h}_e(k) + h_o(k-m)\bar{h}_o(k) = \delta(m), \tag{34}$$

$$\sum_{k=0}^{M-1} g_e(k-m)\bar{h}_e(k) + g_o(k-m)\bar{h}_o(k) = 0. \tag{35}$$

Next we take M-length DFTs of (33)-(35) to obtain, after changing $-u$ to u,

$$|\hat{g}_e(u)|^2 + |\hat{g}_o(u)|^2 = 1, \tag{36}$$

$$|\hat{h}_e(u)|^2 + |\hat{h}_o(u)|^2 = 1, \tag{37}$$

$$\hat{g}_e(u)\overline{\hat{h}_e(u)} + \hat{g}_o(u)\overline{\hat{h}_o(u)} = 0. \tag{38}$$

We also claim that

$$\hat{g}_e(u)\overline{\hat{g}_o(u)} + \hat{h}_e(u)\overline{\hat{h}_o(u)} = 0. \tag{39}$$

To see this, consider the left-most column of (3). We find that

$$\sum_{m=0}^{M-1} \bar{g}(n-2m)g(-2m) + \bar{h}(n-2m)h(-2m) = \delta(n), \tag{40}$$

where here δ denotes the N-periodic Kronecker delta. For $n = 2k+1$ and $k = 0, \ldots, M-1$, this becomes

$$\sum_{m=0}^{M-1} \bar{g}_o(k-m)g_e(-m) + \bar{h}_o(k-m)h_e(-m) = 0.$$

Taking M-length DFTs and changing $-u$ to u yields (39).

We now claim that (36)-(39) are equivalent to (3)-(6). It is not hard to see that (36)-(38) are respectively equivalent to (4)-(6). However, (39) alone does not imply (3). The reason for this, as will become clear below, is that we must consider both the even and the odd columns of (3).

Lemma 6: Equations (36)-(39) together imply (3).

Proof: We begin with the following observations. First, it is not hard to show that (36)-(39) imply

$$|\hat{h}_e(u)| = |\hat{g}_o(u)| \quad \text{and} \quad |\hat{h}_o(u)| = |\hat{g}_e(u)|. \tag{41}$$

Substituting (41) into (36) and (37) gives

$$|\hat{g}_e(u)|^2 + |\hat{h}_e(u)|^2 = 1, \tag{42}$$

and

$$|\hat{g}_o(u)|^2 + |\hat{h}_o(u)|^2 = 1. \tag{43}$$

(Conversely, it is also true that (42), (43), (38), and (39) imply (41), and hence (36) and (37) as well.) We now observe that (42) also follows directly from (40) by letting $n = 2k$ and taking M-length DFTs. To obtain (43) directly, consider the second column of (3) (column index 1). We obtain

$$\sum_{m=0}^{M-1} \bar{g}(n-2m)g(1-2m) + \bar{h}(n-2m)h(1-2m) = \delta(n-1). \tag{44}$$

Letting $n = 2k+1$ and taking M-length DFTs yields (43). We can now see that (36)–(39) together imply (41), which then gives us (42) and (43). Then from (39) we recover (40) with $n = 2k+1$; from (42) we recover (40) with $n = 2k$, and from (43) we recover (44) with $n = 2k+1$. Note that (44) with $n = 2k$ is equivalent to (40) with $n = 2k+1$, which we have already recovered. □

Remark: Similar arguments also show that (3) and (6) alone are equivalent (42), (43), (38), and (39). This was the approach used in [2]. With either approach we obtain (41), on which we focus next.

Equation (41) leads us to impose the constraints

$$h_e(u) = \overline{g_o(u)} \quad \text{and} \quad h_o(u) = -\overline{g_e(u)}. \tag{45}$$

Lemma 7: The condition (45) is equivalent to

$$h(n) = (-1)^n \overline{g(1-n)}. \tag{46}$$

Proof: To show that (45) implies (46), substitute (45) into the easily verified formula,

$$h(u) = \hat{h}_e(u) + \hat{h}_o(u)e^{-j\frac{2\pi}{N}u}, \tag{47}$$

where $\hat{h}$ is the N-length DFT of h. We obtain

$$\hat{h}(u) = \overline{\hat{g}_o(u)} - \overline{\hat{g}_e(u)}e^{-j\frac{2\pi}{N}u},$$

from which (46) follows. The fact that (46) implies (45) follows by separately considering (46) with n replaced by $2n+1$ and by $2n$. □

Theorem 8: If G and H are of the form (26) and (27), respectively, then (30) and (46) imply that (3)–(6) hold.

Proof: Since (46) implies (45), we clearly have (38) and (39). Next, recall that (30) is equivalent to (36), and if (36) holds, then (45) implies (37). Finally, since (36)–(39) are equivalent to (3)–(6), the theorem follows. □

Remark: Solutions of (30) are easy to construct because (30) is equivalent to (36). Suppose that $\hat{g}_e(u)$ is arbitrarily given. Without loss of generality, we may assume that $\hat{g}_e(u)$ has been scaled so that $|\hat{g}_e(u)|^2 < 1$ for all u. For each u, let $\hat{g}_o(u)$ be any complex number such that $|\hat{g}_o(u)|^2 = 1 - |\hat{g}_e(u)|^2$. Then (36) holds for all u. If we let $h(n)$ be given by (46), then G and H will satisfy (3)–(6). In the next section, we consider the more delicate task of constructing $g(n)$ so that in addition to (30), we also have $g(n) = 0$ for most values of n.

B. Selection of G_0

As before we drop the subscript 0. The purpose of this subsection is to find a solution of (30) such that $g(n) = 0$ for most values of n.

Let $\lambda(m)$ denote the sum in (30). Clearly, λ has period M. Now observe that as m takes the values $1, \ldots, M/2-1$, the number $M-m$ takes the values $M-1, \ldots, M/2+1$. Thus, $\lambda(m) = 0$ for $m = 1, \ldots, M/2-1$ if and only if $\lambda(m) = 0$ for $m = M/2+1, \ldots, M-1$. Therefore, (30) is equivalent to saying that $\lambda(m) = 0$ for $m = 0, \ldots, M/2$. We now impose the constraint $g(n) = 0$ for $p \le n \le N-1$, where $p \le M$ and p is even. It then follows that (30) can be replaced by

$$\sum_{n=0}^{p-1} g(n)\bar{g}(n+2m) = \delta(m), \quad m = 0, \ldots, M/2. \tag{48}$$

Now, in (48) $n + 2m \le (p-1) + 2(M/2) = p - 1 + M \le N - 1$, since we assumed $p \le M$. So, the sum in (48) will automatically be zero if $2m \ge p$. Thus, we may rewrite (48) as

$$\sum_{n=0}^{p-1} g(n)\bar{g}(n+2m) = \delta(m), \quad m = 0, \ldots, p/2-1. \tag{49}$$

Finally, observe that $n + 2m \le p - 1$ if and only if $n \le p - 2m - 1$, and since $m \le p/2 - 1$, $p - 2m - 1 \ge 1$. Hence, (49) becomes

$$\sum_{n=0}^{p-2m-1} g(n)\bar{g}(n+2m) = \delta(m), \quad m = 0, \ldots, p/2-1. \tag{50}$$

Remark: Obviously, (50) does not depend on N. Now suppose p is even and that we have a solution to (50). Suppose also that N and J are such that $2 \le p \le N/2^J$. Clearly, if g_j is given by (19), then (30) will hold for g_j. If we now define h_j by (20), Theorem 8 shows that (3)–(6) will hold for G_j and H_j if they are defined by (14) and (15), respectively. As argued in Section III-B, we obtain a fast wavelet transform.

We now observe that (50) gives us only $p/2$ nonlinear equations in p unknowns. Therefore, we must impose ad-

ditional consistent constraints on $g(0), \ldots, g(p-1)$ in order to fix a solution. The linear constraints suggested by Daubechies [2] are

$$\frac{d^k}{dz^k}\left(\sum_{n=0}^{p-1} g(n)z^n\right)\Bigg|_{z=-1} = 0 \quad \text{for } k = 0, \ldots, p/2-1. \quad (51)$$

Theorem 9 (Daubechies): The constraints (51) are consistent with (50). In fact, we can require that $g(n)$ be real valued.

The proof of this result, which is effectively contained in [2, Section 4.B.], is sketched in the Appendix.

Remark: The theorem statement only guarantees existence of a solution to (50)-(51). In fact, the proof is constructive, obtaining solutions by factoring certain polynomials; however, this is not the only way to obtain a solution in a given instance. One can attempt a direct solution by hand if the system of equations is small (e.g., $p = 4$) or by numerical techniques such as Newton's method otherwise. However, see also the remark at the end of the Appendix. Note also that if $g(n)$ is real, then so is $h(n)$. Hence, D can also be viewed as a mapping from $\mathbb{R}^N$ to $\mathbb{R}^N$.

C. Interpretation of (51)

We now show that taking $k = 0$ in (51) suggests that G can be viewed as a low-pass filter and that H can be viewed as a high-pass filter. We assume that (3)-(6) hold with G and H given by (26)-(27).

Recalling that $g(n) = 0$ for $n = p, \ldots, N-1$, and taking $k = 0$ in (51) yields

$$\sum_{n=0}^{N-1} g(n)(-1)^n = 0, \quad (52)$$

or, equivalently,

$$\sum_{m=0}^{M-1} g_e(m) = \sum_{m=0}^{M-1} g_o(m).$$

In other words,

$$\sum_{n=0}^{N-1} g(n) = C, \quad (53)$$

where $C \triangleq 2\sum_{m=0}^{M-1} g_e(m)$. If we set $x(n) \equiv 1$, then (53) implies that

$$\sum_{n=0}^{N-1} g(n-2m)x(n) = C\,x(2m). \quad (54)$$

In other words, the operator G simply scales and subsamples the d.c. signal $x(n)$. Hence, we view G as a low-pass filter. Now suppose that we substitute (46) into (52). We find that

$$\sum_{n=0}^{N-1} h(n) = 0, \quad (55)$$

from which it follows that if $x(n) \equiv 1$, then

$$\sum_{n=0}^{N-1} h(n-2m)x(n) = 0.$$

Since H blocks the d.c. signal $x(n)$, we view H as a high-pass filter.

We now show that the constant C in (54) satisfies $C^2 = 2$. For $x(n) \equiv 1$, we just showed that $Hx = 0$. Thus, (3) implies $\|G^*Gx\|^2 = \|x\|^2$. Next, (4) implies $\|G^*Gx\|^2 = \|Gx\|^2$. Finally, $\|x\|^2 = N$, and by (54), $\|Gx\|^2 = C^2N/2$.

For $k \geq 1$, (51) simply specifies how G and H process certain higher-variation signals.

Remark: Daubechies' original reason for introducing (51) was related to regularizing the behavior of the continuous-time analog of (18) for large J [2, Section 3.B.].

Appendix
Proof of Theorem 9

The proof is self-contained except for references to [2, Lemmas 4.2 and 4.4]. Since these lemmas are statements about certain polynomials, their proofs in [2] can be read independently of the rest of [2].

With slight abuse of notation, let $G(z)$ denote the polynomial appearing in (51). Then (51) is equivalent to saying that $(1+z)^{p/2}$ is a factor of $G(z)$, and hence,

$$G(z) = [\tfrac{1}{2}(1+z)]^{p/2}F(z) \quad (56)$$

for some polynomial F of degree $p/2-1$. The factor of $1/2$ is included for convenience in the derivation below. Observe that the coefficients of G are all real if and only if the coefficients of F are all real. We now require that $g(n)$ be real. Thus, it suffices to prove the existence of a polynomial F of degree $p/2-1$ with real coefficients such that if $G(z)$ is defined by (56), then (50) holds.

To begin, first recall that (50) was derived from (30), and (30) is equivalent to (36). If we let $\hat{g}(u)$ denote the N-length DFT of $g(n)$ and use the analog of (47), we find that (36) is equivalent to

$$|\hat{g}(u)|^2 + |\hat{g}(u+N/2)|^2 = 2. \quad (57)$$

Now observe that $\hat{g}(u) = G(e^{j\xi})$ when $\xi = -2\pi u/N$ and u is an integer. If (56) holds, then

$$|G(e^{j\xi})|^2 = [\cos^2(\xi/2)]^{p/2}|F(e^{j\xi})|^2.$$

Next, if $F(z)$ has real coefficients, then $|F(e^{j\xi})|^2$ also has real coefficients and is real-valued. Hence, we can write $|F(e^{j\xi})|^2 = F(e^{j\xi})F(e^{-j\xi})$ as a polynomial of degree $p/2-1$ in $\cos\xi$, or equivalently, in $\sin^2(\xi/2)$; i.e., we may write $|F(e^{j\xi})|^2 = P(\sin^2(\xi/2))$, where P is a polynomial of degree $p/2-1$. So, if (56) holds for some F with real coefficients, and if (57) holds (which is what we are trying to prove), then

$$y^{p/2}P(1-y) + (1-y)^{p/2}P(y) = 2, \qquad (58)$$

where $y \triangleq \cos^2(\xi/2)$, $\xi \triangleq -2\pi u/N$, and u is an integer. Conversely, in [2, p. 975, Lemma 4.4] it is proved that the polynomial

$$P(y) \triangleq 2\sum_{k=0}^{p/2-1}\binom{p/2-1+k}{k}y^k$$

satisfies (58). Furthermore, $P(y)$ is nonnegative for $0 \le y \le 1$, and by [2, p. 972, Lemma 4.2], there exists a (nonunique) polynomial Q of degree $p/2-1$ with real coefficients that satisfies $|Q(e^{j\xi})|^2 = P(\sin^2(\xi/2))$. Thus, if we take the Q of Daubechies and obtain the $g(n)$ from

$$G(z) = [\tfrac{1}{2}(1+z)]^{p/2}Q(z),$$

and set $\xi = -2\pi u/N$ and $y = \cos^2(\xi/2)$, then

$$\begin{aligned}|g(u)|^2 + |g(u+N/2)|^2 &= |G(e^{j\xi})|^2 + |G(e^{j(\xi-\pi)})|^2 \\ &= y^{p/2}P(1-y) + (1-y)^{p/2}P(y) \\ &= 2.\end{aligned}$$

□

Remark: In general there are $p/2$ choices for $Q(z)$ (and hence $G(z)$). The coefficients $g(n)$ (called $h(n)$ in [2]) given in [2, Table 1, p. 980] correspond to taking $Q(z)$ to have all its roots inside the unit circle; i.e., $Q(z)$ is a minimum-phase filter.

References

[1] P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a compact image code," *IEEE Trans. Commun.*, vol. COM-31, pp. 532-540, Apr. 1983.

[2] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Commun. Pure Appl. Math.*, vol. 41, pp. 909-996, Nov. 1988.

[3] ——, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Trans. Inform. Theory*, vol. 36, no. 5, pp. 961-1005, Sept. 1990.

[4] I. Gohberg and S. Goldberg, *Basic Operator Theory*. Boston: Birkhäuser, 1980.

[5] C. E. Heil and D. F. Walnut, "Continuous and discrete wavelet transforms," *SIAM Review*, vol. 31, no. 4, pp. 628-666, Dec. 1989.

[6] F. Hlawatsch and G. F. Boudreaux-Bartels "Linear and quadratic time-frequency signal representations," *IEEE Signal Proc. Mag.*, vol. 9, no. 2, pp. 21-67, Apr. 1992.

[7] K. Hoffman and R. Kunze, *Linear Algebra*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[8] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 7, pp. 674-693, July 1989.

[9] ——, "Multiresolution approximations and wavelet orthonormal bases of $L^2(\mathbb{R})$," *Trans. Am. Math. Soc.*, vol. 135, no. 1, pp. 69-88, Sept. 1989.

[10] ——, "Multifrequency channel decompositions of images and wavelet models," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 37, no. 12, pp. 2091-2110, Dec. 1989.

[11] Z. J. Mou and P. Duhamel, "Short-length FIR filters and their use in fast nonrecursive filtering," *IEEE Trans. Signal Proc.*, vol. 39, pp. 1322-1332, June 1991.

[12] O. Rioul and P. Duhamel "Fast wavelet algorithms for discrete and continuous wavelet transforms," *IEEE Trans. Inform. Theory*, vol. 38, no. 2, pp. 569-586, Mar. 1992.

[13] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Proc. Mag.*, vol. 8, no. 4, pp. 14-38, Oct. 1991.

[14] M. J. T. Smith and T. P. Barnwell III, "Exact reconstruction techniques for tree-structured subband coders," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-34, no. 3, pp. 434-441, June 1986.

[15] G. Strang, "Wavelets and dilation equations: A brief introduction," *SIAM Review*, vol. 31, no. 4, pp. 614-627, Dec. 1989.

[16] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-36, pp. 730-738, May 1988.

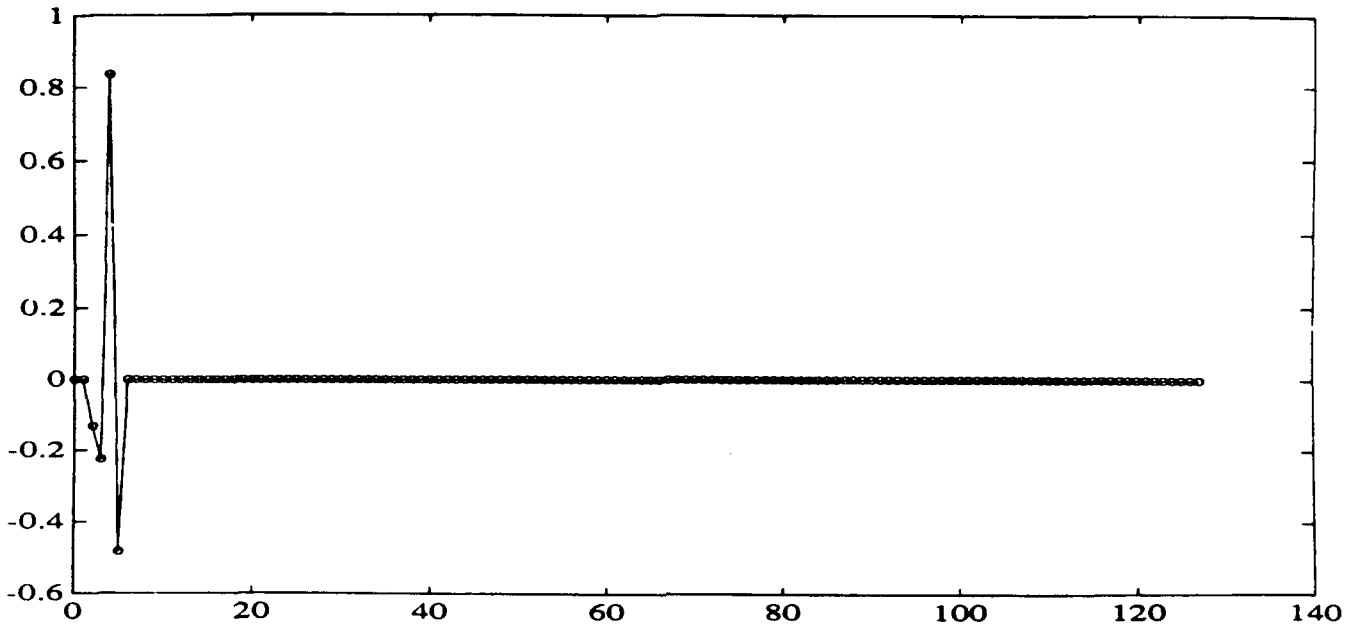


Fig. 1. Plot of $\psi_{1,2}(n)$ from Example 1.

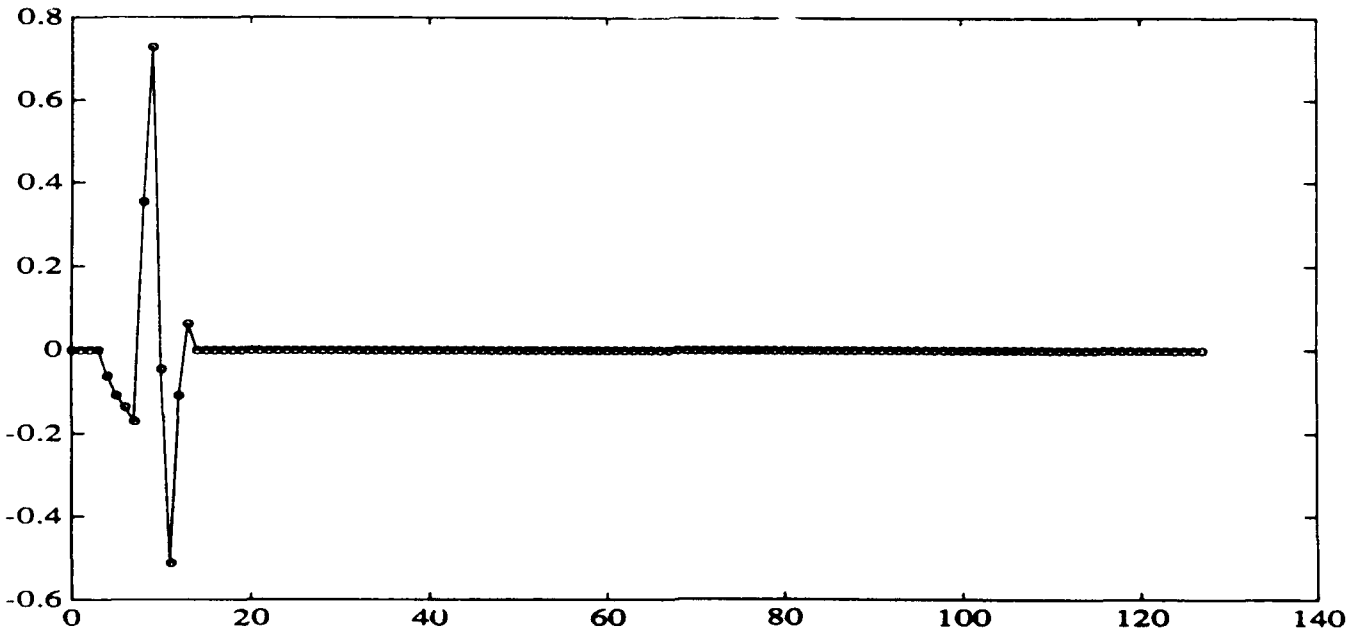


Fig. 2. Plot of $\psi_{2,2}(n)$ from Example 1.

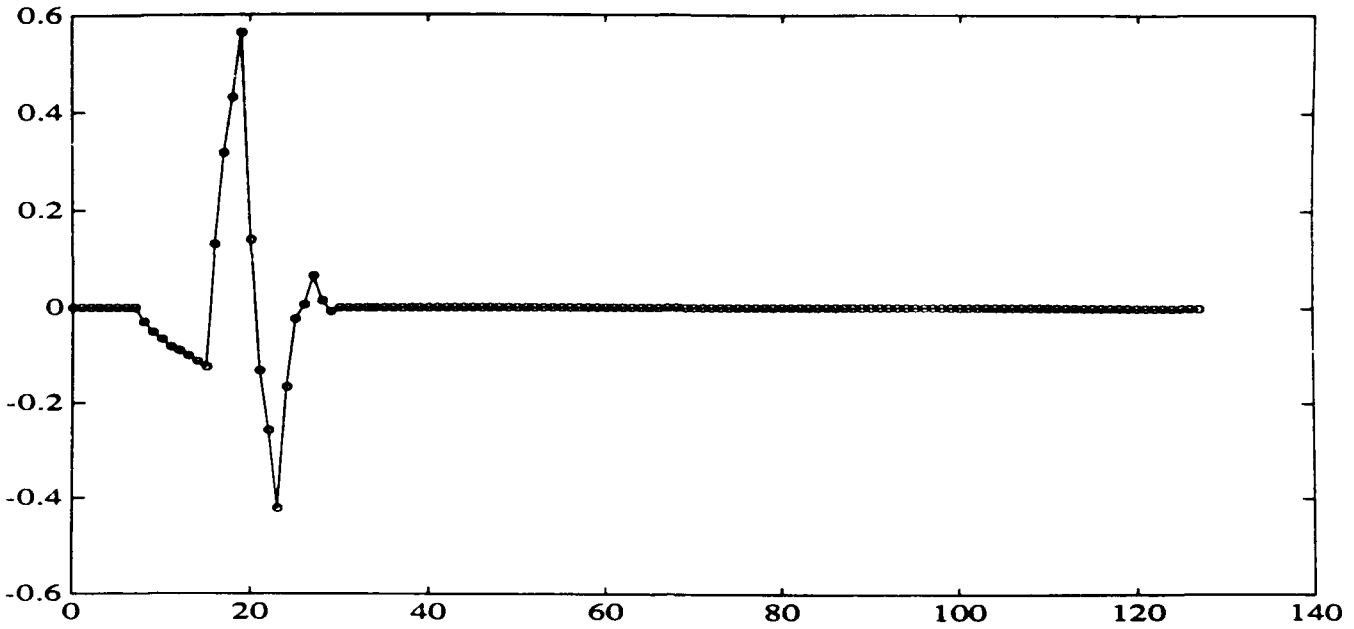


Fig. 3. Plot of $\psi_{3,2}(n)$ from Example 1.

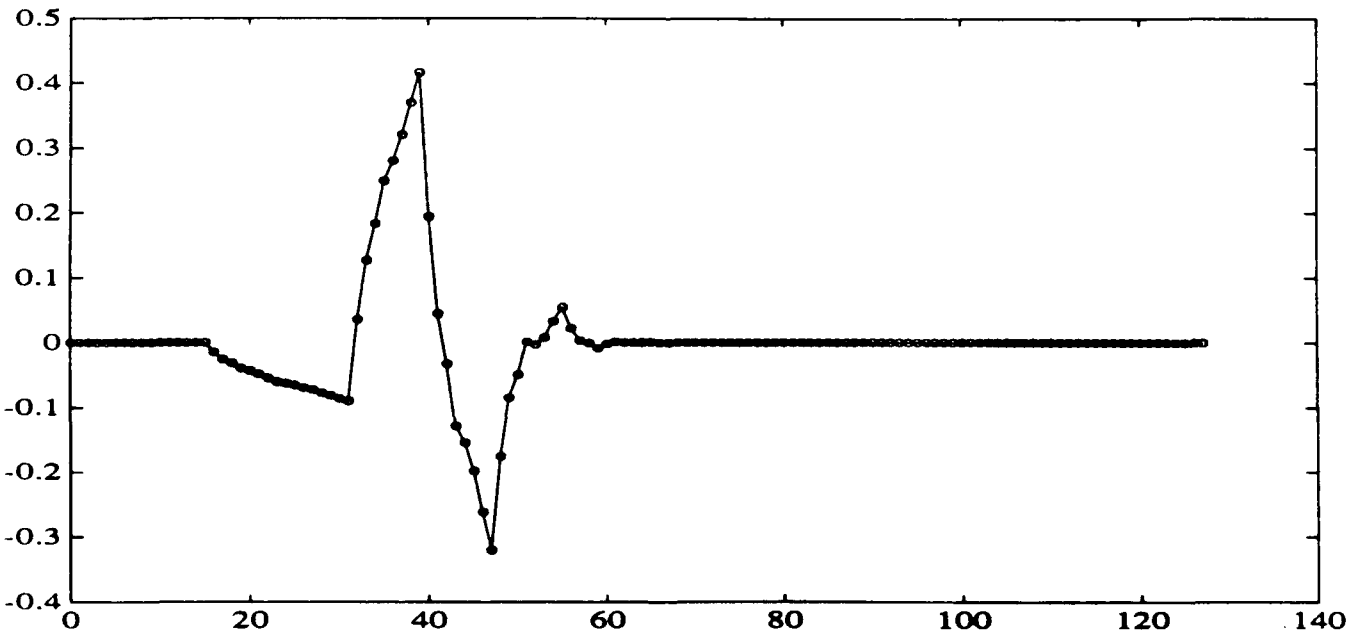


Fig. 4. Plot of $\psi_{4,2}(n)$ from Example 1.

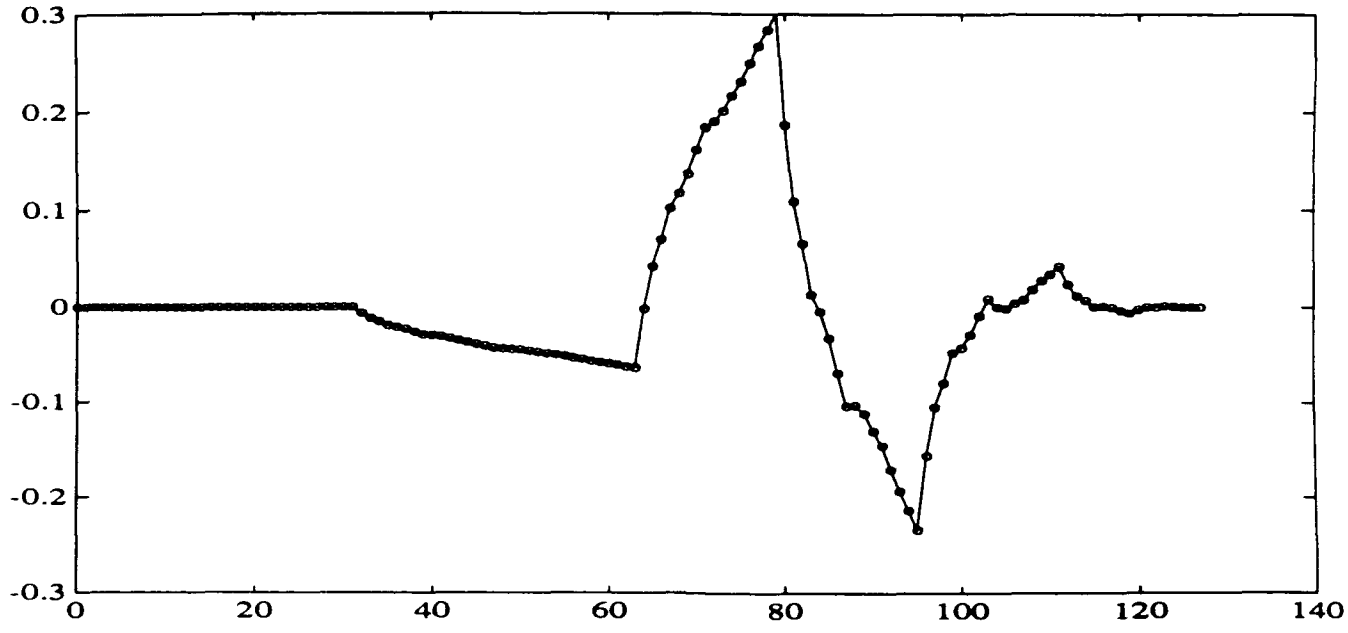


Fig. 5. Plot of $\psi_{5,2}(n)$ from Example 1.

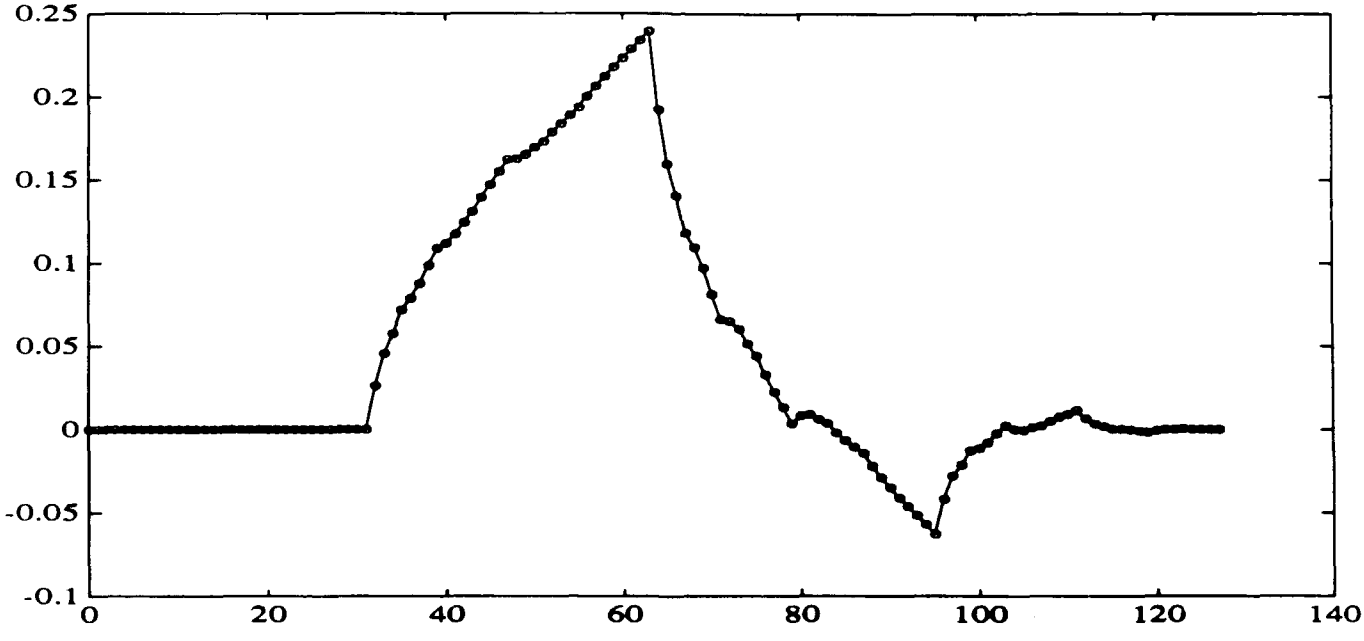


Fig. 6. Plot of $\varphi_{5,1}(n)$ from Example 1.

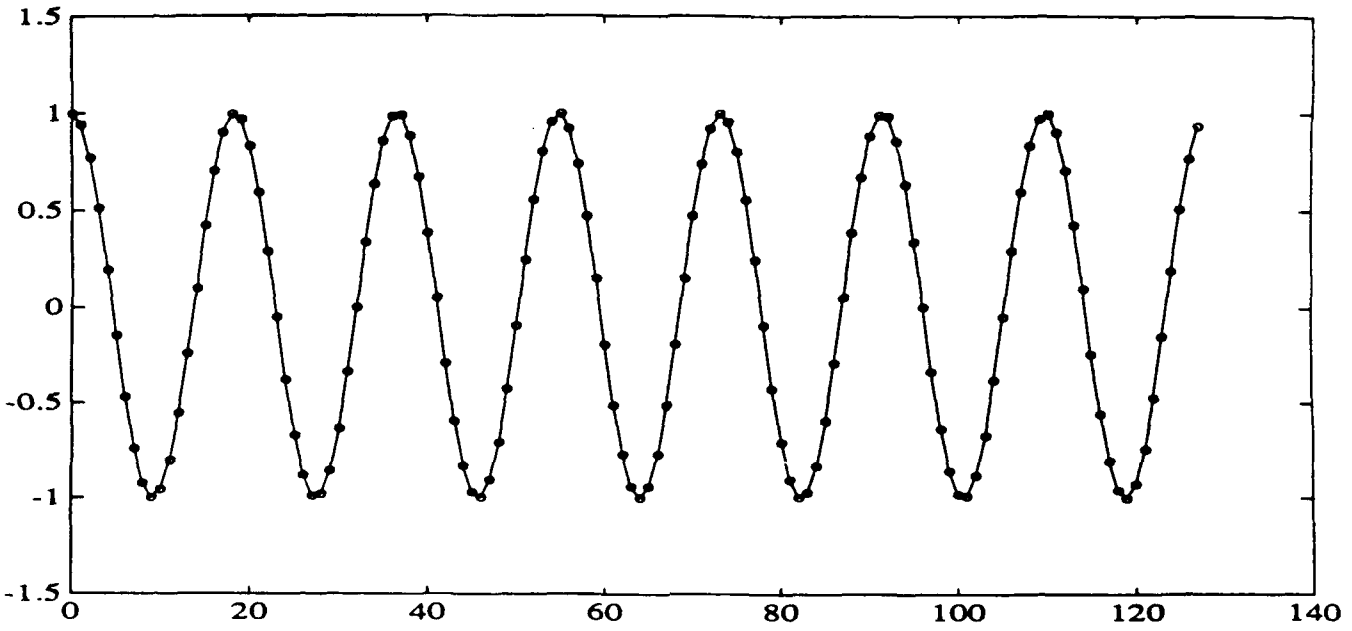


Fig. 7. Graph of $x_0(n)$ defined in (22).

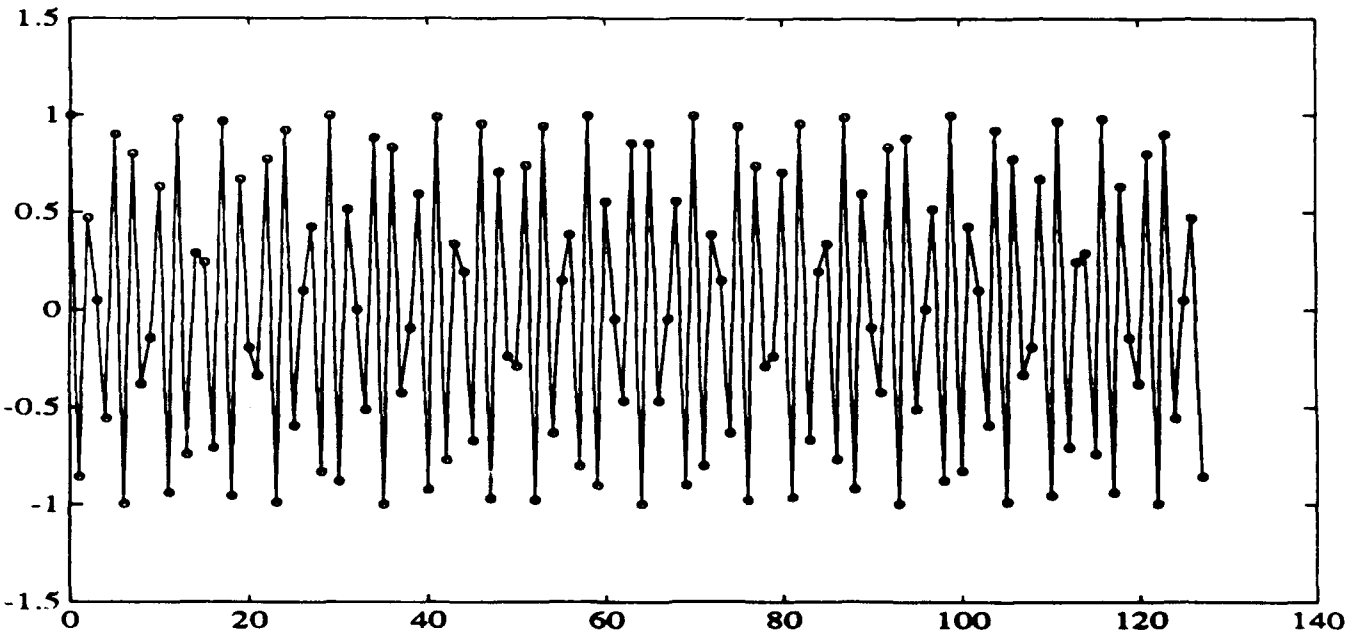


Fig. 8. Graph of $x_0(n)$ defined in (23).

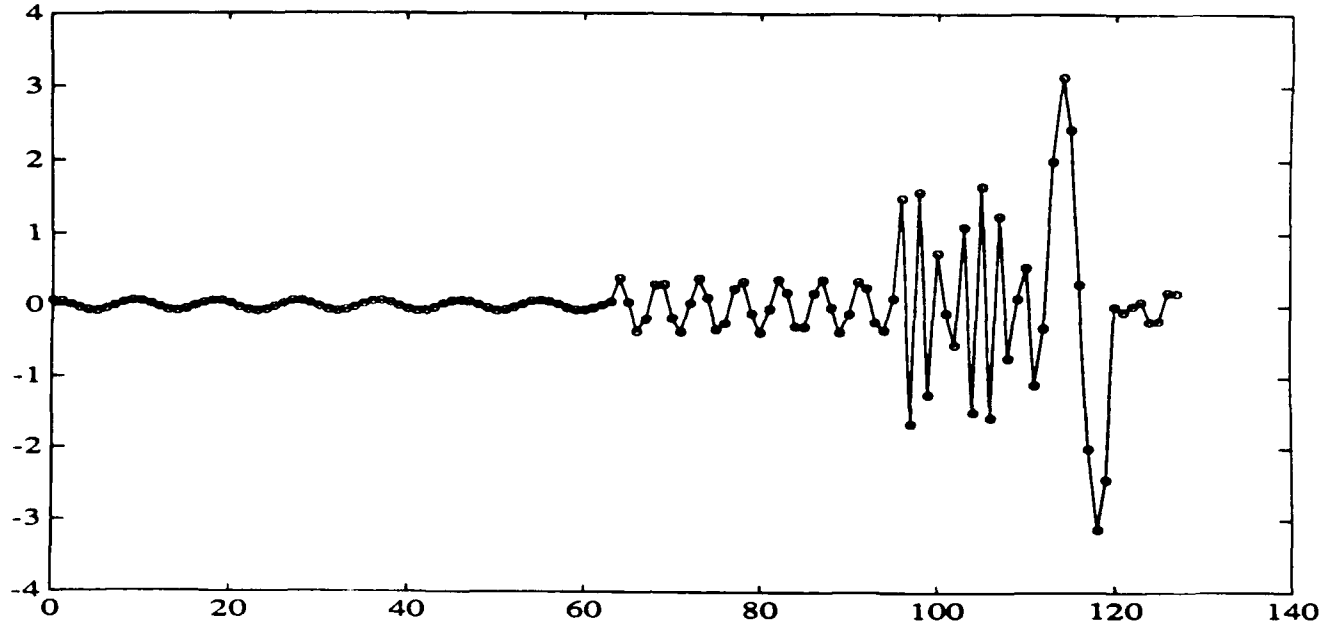


Fig. 9. DWT of Fig. 7.

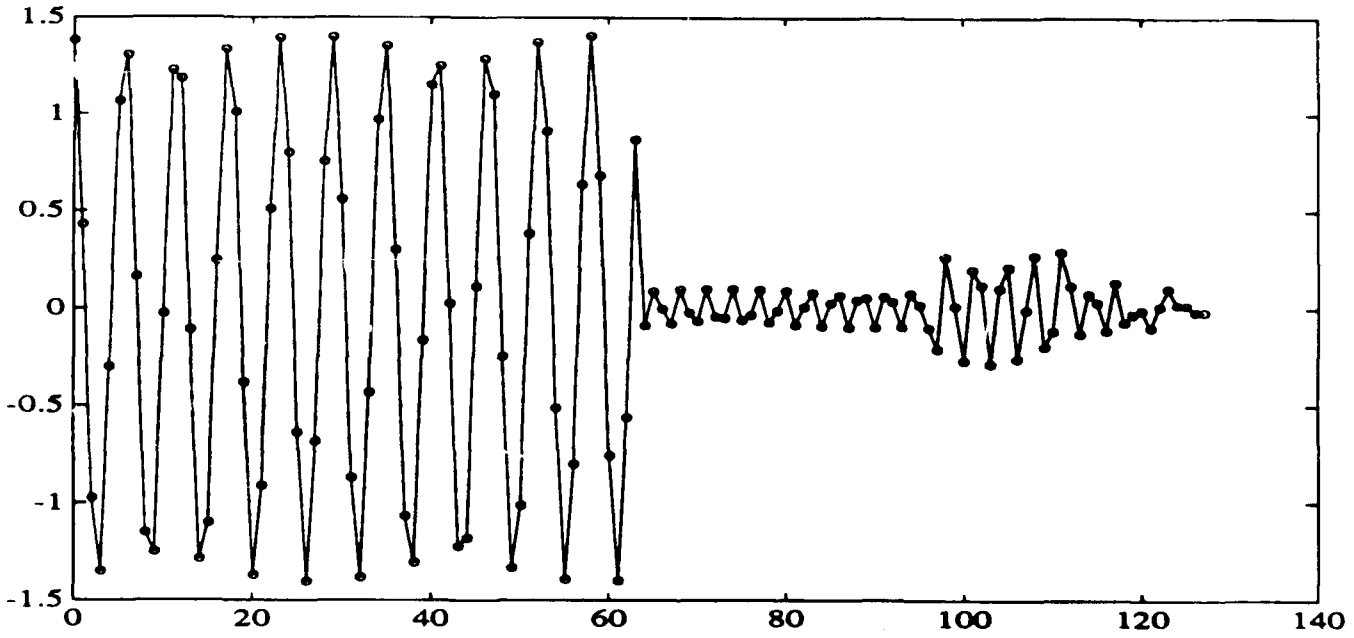


Fig. 10. DWT of Fig. 8.

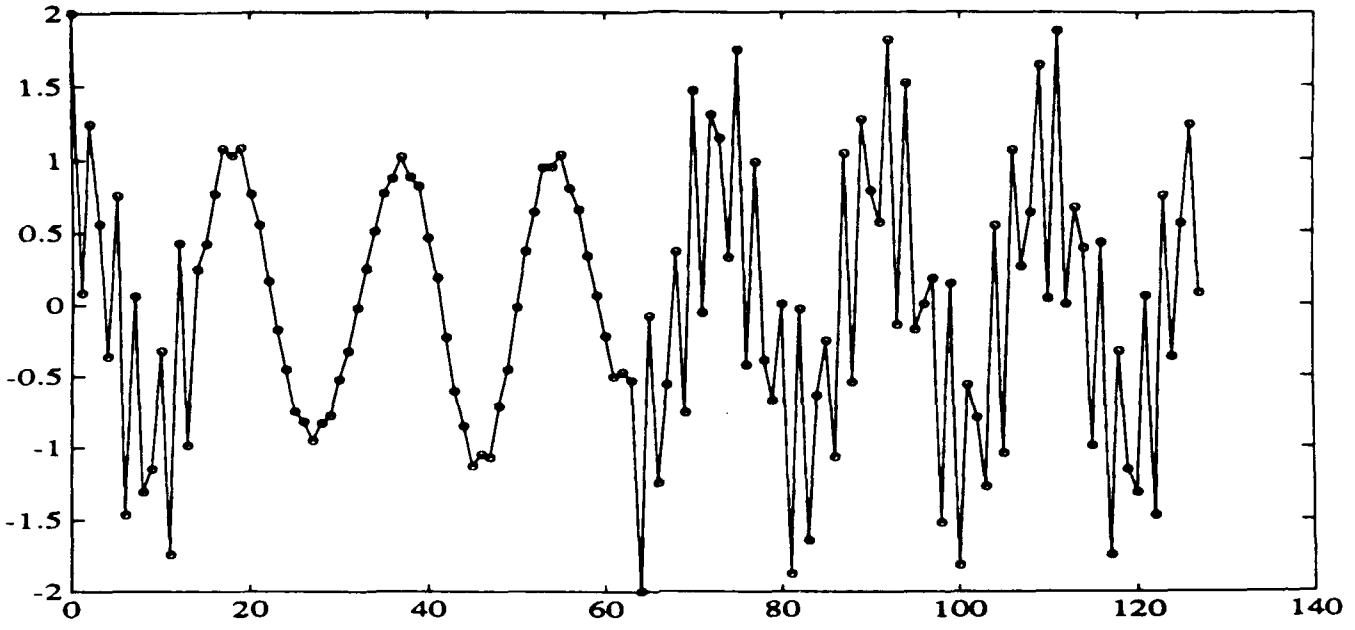


Fig. 11. Time-frequency filtered signal of Example 3.

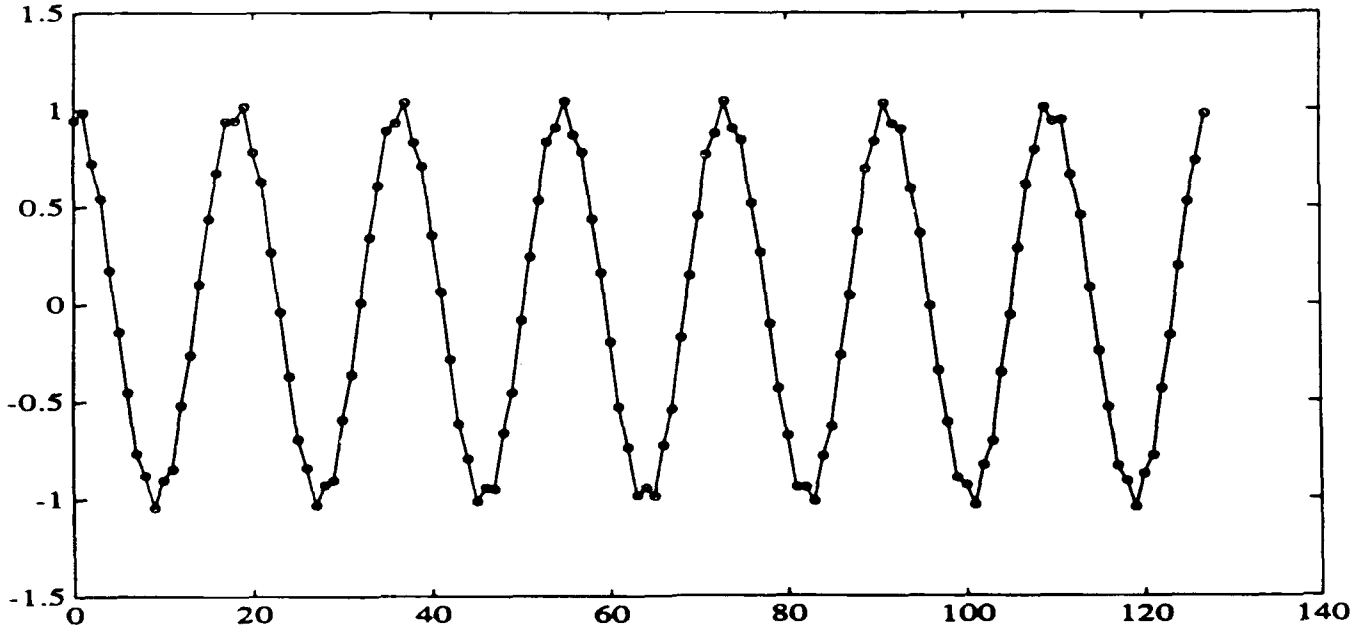


Fig. 12. Reconstruction after compression of signal in Fig. 7.

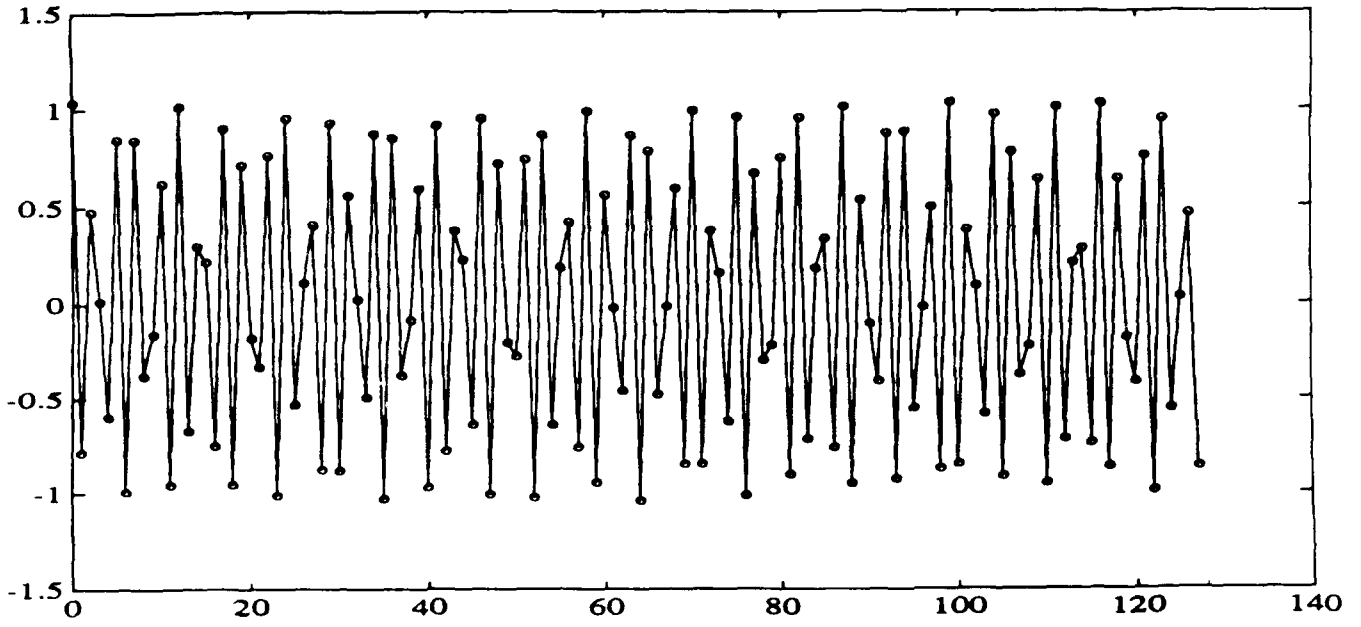


Fig. 13. Reconstruction after compression of signal in Fig. 8.

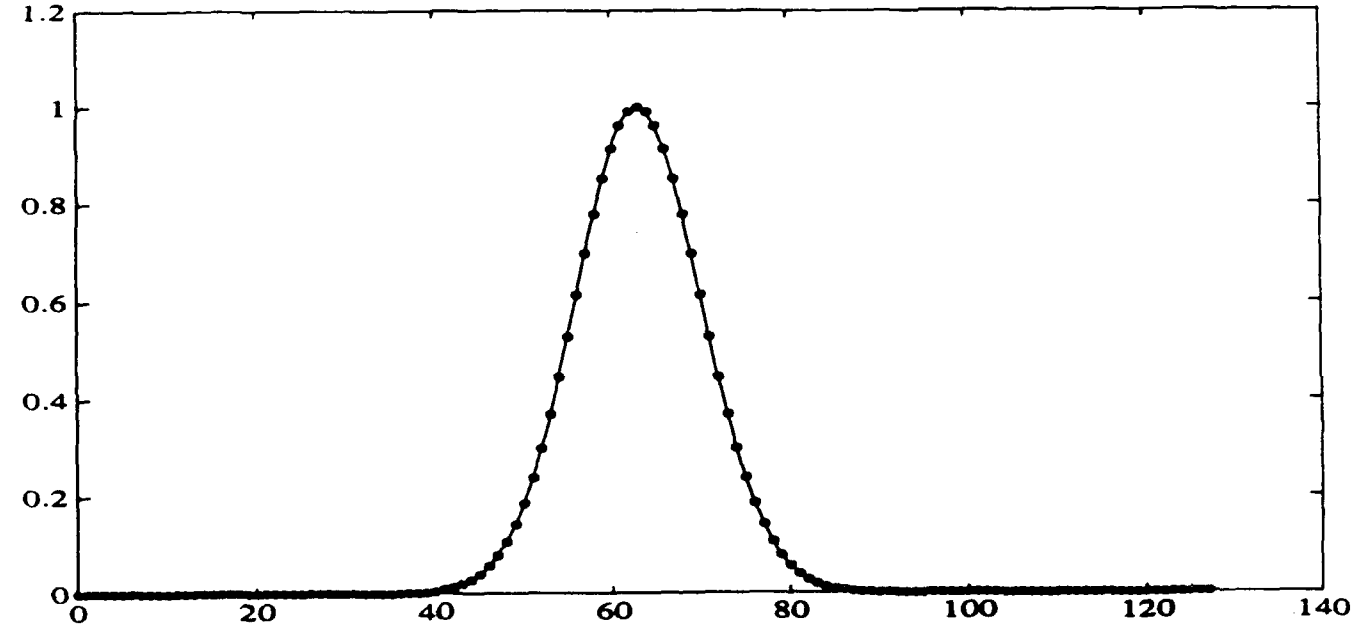


Fig. 14. Graph of $x_0(n)$ defined in (24).

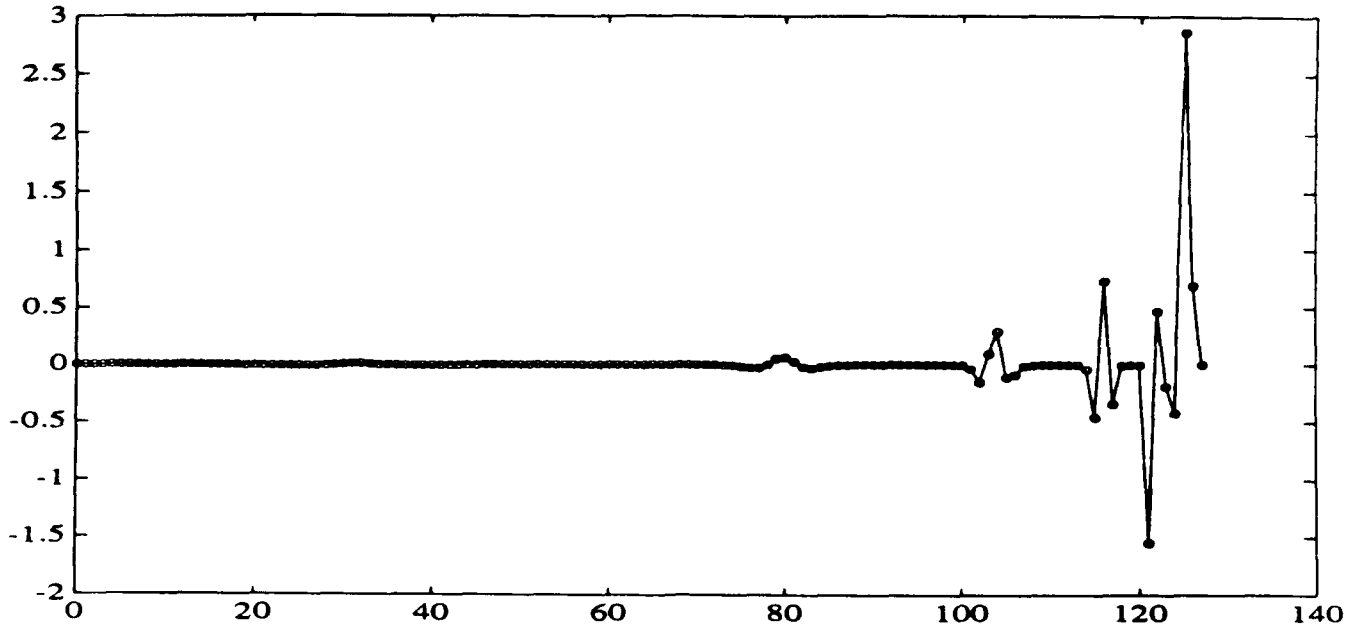


Fig. 15. DWT of Fig. 14.

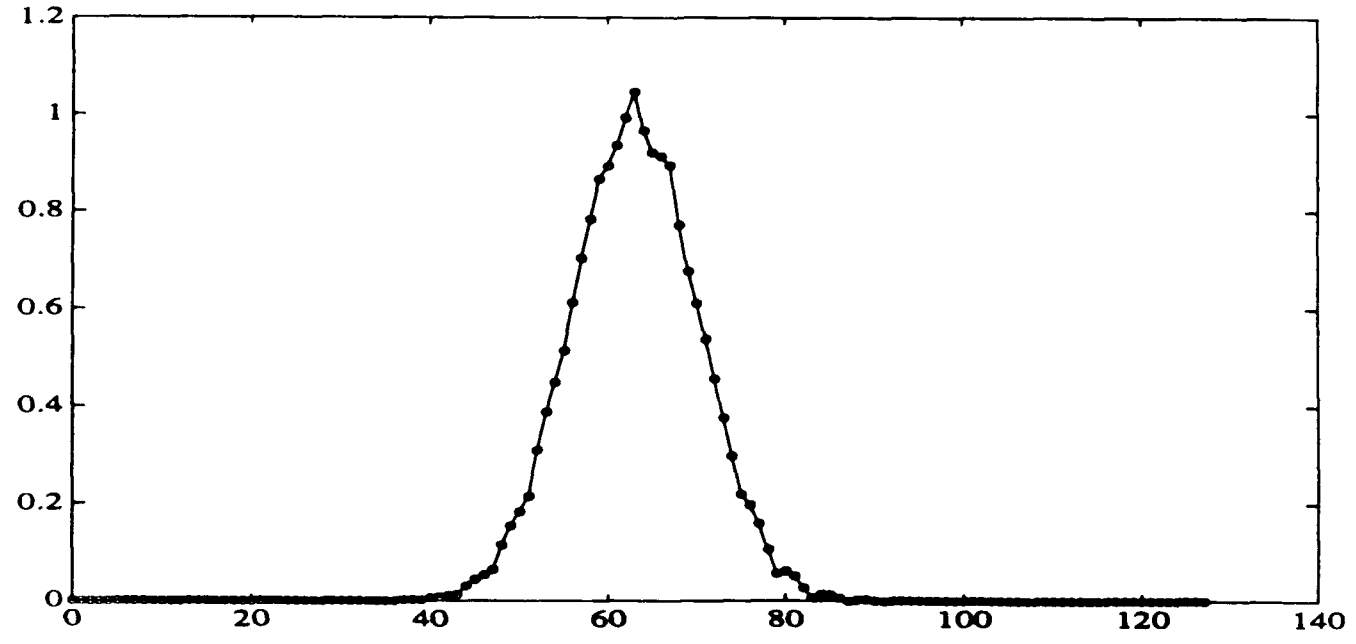


Fig. 16. Reconstruction after compression of signal in Fig. 14.